SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

DIPLOMSKI RAD br. 1781

# Nenadzirano učenje procjene dubine jednookim vidom

Kristijan Bartol

Zagreb, ožujak 2019.

*Umjesto ove stranice umetnite izvornik Vašeg rada.*
*Da bi ste uklonili ovu stranicu obrišite naredbu* `\izvornik`.

# SADRŽAJ

# 1. Uvod

Računalni vid područje je računarske znanosti koje se bavi razumijevanjem scene obradom slike ili niza slika iz sustava kamera. Algoritmima za analizu niza slika je, između ostalog, moguće odrediti vlastito gibanje, informaciju koja se sastoji od vlastitog položaja (točke) i pomaka u odnosu na položaj u prethodnoj slici (vektor) te udaljenost drugih (na slikama vidljivih) objekata scene od sustava kamera. Opisano znanje o sceni suštinsko je za mnoge algoritme koji se oslanjaju na vid. Postoje razne tehnike izlučivanja znanja iz slika, a mogu se podijeliti na algoritme dubokog učenja i ostale, "klasične" algoritme.

Duboko učenje računalnom vidu, kao i sve brojnijim drugim područjima, nudi efikasna i točna rješenja na neke dotad vrlo teške probleme analize slike. Valja istaknuti mjeru u kojoj je računalni vid preobražen dubokim učenjem. Ona se jasno može vidjeti iz popisa odabranih članaka na jednoj od najvećih konferencija računalnog vida - CVPR-a [1]. Modifikacijama osnovnih arhitektura za klasifikaciju slika, uz manje prilagodbe, moguće je dubinski rekonstruirati scenu, što je opisano u ovom radu.

U prvom poglavlju rada opisani su uvodni teoretski pojmovi geometrije slike i točkastog modela kamere. Pokazana je formalna veza između geometrijskih pojmova: dubine, oblaka točaka i vlastitog gibanja. Uvedena je intrinzična i ekstrinzična matrica točkastog modela.

U drugom poglavlju uvode se osnovni pojmovi strojnog učenja kroz primjere jednostavnih klasifikacijskih i regresijskih hipoteza koje vode do poopćenog linearnog modela. Mreža unaprijedno povezanih poopćenih linearnih modela je unaprijedni model. Opisana je propagacija gradijenata unatrag kao postupak učenja unaprijednog modela. Konačno, opisan je konvolucijski model u kontekstu modernog dubokog učenja i dani su primjeri dubokih arhitektura.

U trećem poglavlju opisan je model dubokog učenja za procjenu dubine i vlastitog gibanja. Istaknuta je temeljna ideja. Opisane su građevne jedinice za procjenu dubine i vlastitog gibanja. Definirana je funkcija gubitka kao zbroj rekonstrukcijske pogreške, gubitka glatkoće mape, strukturne sličnosti originalne i rekonstruirane slike te pogre-

ške iterativne metode najbližih točaka. Detaljno je opisana iterativna metoda najbližih točaka kao i njena uloga u kontekstu modela.

U četvrtom poglavlju istaknuti su poznati skupovi podataka nad kojima je učen model. Opisan je postupak pripreme slika za učenje, proširivanje skupa za učenje te predobrada slika.

U petom poglavlju opisane su komponente implementacije specifične za model iz četvrtog poglavlja i istaknuta jedna od bitnih značajki TensorFlow-a - statički računski graf i diferencijabilno programiranje. Dan je i detaljno opisan izvorni kod za nadjačavanje gradijenata u TensorFlow-u. Konačno, detaljno je opisano nadjačavanje gradijenata čvora iterativne metode najbližih točaka.

U posljednjem poglavlju prikazani su rezultati učenja modela nad skupom podataka KITTI. Odabrana su dva modela za usporedbu. Razlikuju se u korištenoj funkciji pogreške glatkosti procijenjene dubinske mape. Modeli se uspoređuju kvalitativno nad skupom slika te kvantitativno koristeći poznate metrike za evaluaciju rezultata dubinskih mapa. Metrike i postupak evaluacije također su opisani.

Postupak dubokog učenja opisan u ovom radu je nenadziran - koriste se nizovi slika iz skupa KITTI bez znanja o stvarnim dubinama. Ovakav pristup je zanimljiv jer je pribavljanje oznaka dugotrajan i skup zadatak.

# 2. Geometrijske značajke slika

Slika je dvodimenzionalna projekcija stvarnog svijeta (funkcija $f_{slika}$ u 2.4). Zbog nesavršenosti postupka dolazi do gubitka informacije.

$$f_{slika}: \mathbb{R}^3 \rightarrow \mathbb{R}^2 \tag{2.1}$$

Može se reći da je cilj računalnog vida maksimalno iskoristiti informaciju dostupnu sa slike za rekonstrukciju originalne informacije o sceni. Informacije slike dijele se u dvije kategorije značajki: fotometrijske i geometrijske značajke. Fotometrijske značajke su boja piksela i njen intenzitet, a geometrijske su točke, linije i krivulje i njihove koordinate u referentnom koordinatnom sustavu (npr. koordinatnom sustavu lijeve kamere u sustavu dviju kamera). U kontekstu geometrijskih značajki temeljni pojam je *dubina* piksela koja kaže koliko je ono što piksel prikazuje udaljeno od promatrača.

## 2.1. Model kamere

Kamera je uređaj za dobivanje slike - dvodimenzionalne projekcije stvarnog svijeta u analognom ili digitalnom formatu. Preslikavanje je moguće opisati izrazima koji ovise o raznim parametrima na putu od scene do slikovnog platna kamere. Skup izraza koji opisuju preslikavanje naziva se *modelom*.

### 2.1.1. Točkasti model

Kako svijet "natjerati" u dvije dimenzije? Ideja je da svjetlost iz svijeta zabilježimo na slikovnom "platnu" - tankom fizičkom sloju pravokutnih dimenzija koji detektira valnu duljinu fotona koji djeluje s dijelom platna. Problem koji se uočava s pripadne slike 2.1a je taj da će svaka točka stvarnog svijeta pripasti svakoj točki na platnu. Takvo preslikavanje uništilo bi svu informaciju o originalnoj sceni.

**(a)** Preslikavanje svijeta na platno.

**(b)** Preslikavanje na platno kroz rupicu. Slika je umanjena i obrnuta.

**Slika 2.1:** Točkasti model kamere.

Između scene i platna stoga se postavlja pregrada s rupicom u sredini kao na slici 2.1b. Sada se prethodni efekt umanjio i pravilnim odabirom promjera rupice dobiva se čista slika. Ovakav se model kamere naziva točkasti model (eng. *pinhole camera model*). Dobivena slika je umanjena, obrnuta i realna.

Kako bi jednostavnije postavili geometriju slike, uvodi se virtualno platno koje se postavlja između prepreke i svijeta. Prepreka se sad naziva *kamera* (slika 2.2). U koordinatnom sustavu kamere, označe se X, Y i Z osi. Udaljenost od središta koordinatnog sustava (centra kamere) označenog s $C$ do virtualnog platna je *fokalna daljina* i označi se s $f$, a točka u kojoj ta normala siječe platno naziva se *centar slike* i označen je s $c$. *Projekcijska linija* povezuje centar kamere i točku stvarnog svijeta koja se projicira na platno. Originalna točka označena je s $P$, a točka u kojoj linija siječe virtualno platno je $p$.

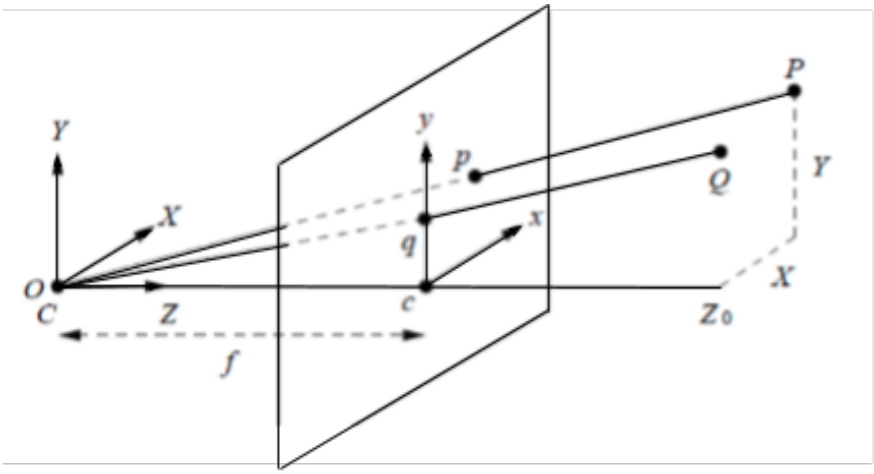
Originalna točka svijeta je trodimenzionalna $P = (X, Y, Z)^T$, a projicirana je dvodimenzionalna $p = (x, y)^T$. Iz simetrije trokuta $\Delta Ocp$ i trokuta $\Delta OZP$ sa slike 2.2 dobiva se

$$\frac{X}{Z} = \frac{x}{f}, \tag{2.2}$$

$$\frac{Y}{Z} = \frac{y}{f} \tag{2.3}$$

odnosno,

$$x = f\frac{X}{Z} \tag{2.4}$$




**Slika 2.2:** Označene udaljenosti kod točkastog modela kamere.

$$y = f\frac{Y}{Z}. \tag{2.5}$$

Ovo je najjednostavnija forma *perspektivne projekcije*. U pravilu, udaljenosti između stvarnih predmeta znatno su veće od udaljenosti između piksela koji te udaljenosti prikazuju (slika 2.2). Uz pretpostavku da su sve točke scene jednako udaljene od kamere ($Z$ svima jednak, označi se sa $Z_0$), uvodi se *faktor skaliranja* $s$:

$$s = \frac{f}{Z_0} \text{ (za svaku točku scene)} \tag{2.6}$$

$$x = sX \tag{2.7}$$

$$y = sY. \tag{2.8}$$

### 2.1.2. Intrinzični parametri

Pikseli senzora kamere nisu savršeno okrugli pa se jednakostima 2.4 i 2.5 dodaju dodatni skalarni parametri $k$ i $l$.

$$x = kf\frac{X}{Z}, \tag{2.9}$$

$$y = lf\frac{Y}{Z}. \tag{2.10}$$

Skalarni umnošci s početaka jednakosti 2.9 i 2.10 mogu se skratiti u skalare $f_x = kf$ i $f_y = lf$. Centar slike $c$ ne mora uvijek biti u istoj referentnoj točki (npr. središtu slikovnog platna) pa se odmak od referentne točke obilježava pomacima $c_x$ i $c_y$. Projekcije nad osima zadane sada su zadane s 2.11 i 2.12:

$$x = f_x \frac{X}{Z} - s\frac{Y}{Z} + c_x, \tag{2.11}$$

$$y = f_y \frac{Y}{Z} + c_y. \tag{2.12}$$

Navedene jednakosti projiciraju točku iz stvarnog svijeta $P = (X, Y, Z)$ na dvodimenzionalno platno u koordinatnom sustavu kamere (za razliku od ekstrinzičnih jednakosti u nastavku). Izlučivši $\frac{1}{Z}$, koristeći homogene koordinate $\widetilde{x} = [x, y, 1]^\top$ te zapisavši sustav jednadžbi oblik u matričnom obliku, dobiva se karaktestični oblik intrinzične matrice (3.7):

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.13}$$

### 2.1.3. Ekstrinzični parametri

Intrinzične jednakosti preslikavaju točku scene koordinatnog sustava kamere u dvodimenzionalno slikovno platno, a ekstrinzične transformiraju točku iz koordinatnog sustava svijeta u koordinatni sustav kamere. Ekstrinzično preslikavanje je, dakle, funkcija

$$f_{ekstr} \colon \mathbb{R}^3 \to \mathbb{R}^3. \tag{2.14}$$

Ekstrinzična transformacija općenito je transformacija *krutog tijela* koja se sastoji od rotacije i translacije. Jednostavno, ekstrinzična matrica govori o tome gdje se kamera nalazi u sceni i u kojem smjeru "gleda". Ekstrinzična se matrica jednostavno može zapisati:

$$[R|t] = \left[\begin{array}{ccc|c} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \end{array}\right] \tag{2.15}$$

Uobičajeno je da se ekstrinzičnoj matrici dodaje 4. redak $[0, 0, 0, 1]^T$ kako bi postala kvadratna. Matrica se sada može rastaviti na rotacijsku i translacijsku:

$$\left[\begin{array}{c|c} R & t \\ \hline 0 & 1 \end{array}\right] = \left[\begin{array}{c|c} I & t \\ \hline 0 & 1 \end{array}\right] \times \left[\begin{array}{c|c} R & 0 \\ \hline 0 & 1 \end{array}\right] \tag{2.16a}$$

$$= \left[\begin{array}{ccc|c} 1 & 0 & 0 & t_1 \\ 0 & 1 & 0 & t_2 \\ 0 & 0 & 1 & t_3 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \times \left[\begin{array}{ccc|c} r_{1,1} & r_{1,2} & r_{1,3} & 0 \\ r_{2,1} & r_{2,2} & r_{2,3} & 0 \\ r_{3,1} & r_{3,2} & r_{3,3} & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{2.16b}$$

Matrica čitave projekcije koja uključuje transformaciju iz koordinatnog sustava svijeta u koordinatni sustav kamere te projekciju transformiranih točaka na platno kamere dobiva se umnoškom intrinzične i ekstrinzične matrice (2.17).

$$P = K \times [R|t] = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \times \left[\begin{array}{ccc|c} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \end{array}\right] \tag{2.17}$$

Ekstrinzična matrica usko je vezana uz matricu vlastitog gibanja, što je detaljno opisano u odsječku 2.3.

### 2.1.4. Geometrijska kalibracija

Geometrijska kalibracija[1] postupak je određivanja parametara kamere, u prvom redu intrinzičnih parametara. Intrinzični parametri su svojstveni kameri i idealno ih se računa samo jednom, prilikom inicijalne kalibracije kamere. Zašto intrinzični parametri nisu svojstveni točkastom modelu, nego kameri? Svaka fizička kamera ima leću kroz koju se svjetlost konačno preslikava na platno, u pripadne piksele. Leća uzrokuje distorziju slike. Kao rezultat, linije koje su u stvarnosti ravne i paralelne na dobivenoj slici izgube ta svojstva (slika 2.3). Ispravljanje distorzije određivanjem kalibracijskih parametara radi se *Zhanovim algoritmom* [2] koji je implementiran u programskim paketima za kalibraciju kamere. Geometrijska kalibracija kamere nužan je korak za pouzdano korištenje kamere u svrhu izlučivanja informacije o sceni.

## 2.2. Dubina

Dubina $d_{ij}$ kazuje koliko je promatrač udaljen od točke scene prikazane pikselom $(i, j)$. Matrica $D_{ij}$ naziva se gusta dubinska mapa ako sadrži informacije o dubini

[1] za razliku od geometrijske, postoji i postupak kalibracije boja, vezan uz fotometrijske značajke

**Slika 2.3:** Primjer distorzije slike na šahovskoj ploči koja se najčešće koristi za geometrijsku kalibraciju kamere. Ravne i paralelne linije u stvarnosti ispadnu zaobljene na slici. Izvor: [35]

svakog piksela slike $I_{ij}$. Rijetka dubinska mapa sadrži informacije o podskupu piksela. U slučaju dovoljnog broja točaka za koje se zna dubina $d_{ij}$, interpolacijom je moguće pouzdano rekonstruirati gustu dubinsku mapu iz rijetke (primjer interpolacije dan je u poglavlju 7).

### 2.2.1. Oblak točaka

Projekcijom dubinske mape sa slikovnog platna u koordinatni sustav kamere dobiva se oblak točaka. Dubinska mapa označi se s $D$, intrinzična matrica je $K$, a oblak točaka neka je $Q$. Za piksel $(i, j)$ pripadna točka u koordinatnom sustavu kamere je dana izrazom 2.18 [3],

$$Q^{ij} = D^{ij} K^{-1} [i, j, 1]^{\top} \tag{2.18}$$

gdje je $[i, j, 1]^T$ pripadni vektor s homogenom koordinatom. Oblak točaka koristan je u prikazu dubinske informacije o sceni i u detekciji eventualnih grubih pogrešaka proračuna dubine. Oblak točaka može se dobiti i iz rijetke dubinske mape.

## 2.3. Vlastito gibanje kamere

Dosad su opisani sustavi kamera i značajke jedne slike. U kontekstu slijednog niza slika javljaju se pojmovi vlastitog gibanja i optičkog toka. Vlastito gibanje opisuje kretanje promatrača kroz scenu. Položaj promatrača u sceni još se naziva i *poza*. Ek-

strinzična matrica opisuje transformaciju iz koordinatnog sustava scene u koordinatni sustav promatrača. Poza opisuje obrnutu vezu.

Neka je s $C$ označen centar, a s $R_c$ rotacijska matrica koja opisuje orijentaciju promatrača u koordinatnom sustavu scene. Transformacijska matrica koja opisuje pozu promatrača je stoga $[R_c|C]$ (uz dodatni 4. redak $(0, 0, 0, 1)$). Ovako opisana transformacija inverz je ekstrinzične matrice:

$$\left[\begin{array}{c|c} R_c & C \\ \hline 0 & 1 \end{array}\right] = \left[\begin{array}{c|c} R & t \\ \hline 0 & 1 \end{array}\right]^{-1} \tag{2.19}$$

Vlastito gibanje je transformacija iz prijašnje poze promatrača u trenutku $t-1$ ($T_{t-1}^{-1}$) u njegovu novu pozu u trenutku $t$ ($T_t^{-1}$) (2.20). Indeks $p$ označava projekciju, a indeksi $1$ i $2$ poredak transformacija. Sve su to, na kraju, transformacije krutog tijela i eksplicitna veza među njima nema posebnu važnost.

$$\left[\begin{array}{c|c} R_p & C_p \\ \hline 0 & 1 \end{array}\right] = \left[\begin{array}{c|c} R_2 & t_2 \\ \hline 0 & 1 \end{array}\right]^{-1} \left[\begin{array}{c|c} R_1 & t_1 \\ \hline 0 & 1 \end{array}\right]^{-1} \tag{2.20}$$

# 3. Duboko učenje u računalnom vidu

Duboko učenje danas je jedna od najpopularnijih grana strojnog učenja i nužno je opisati njihov odnos kroz najbitnije pojmove. Poglavlje je organizirano na način da se opisuju modeli strojnog učenja od jednostavnih prema složenijim. Završava se opisom konvolucijskih arhitektura.

## 3.1. Komponente algoritma strojnog učenja

Cilj je algoritma strojnog učenja pronaći hipotezu $h$ koja najbolje pristaje podacima, ovisno o definiranoj funkciji cilja. U prethodnoj rečenici objedinjene su tri osnovne komponente svakog algoritma strojnog učenja:

- model (vezan uz pojam hipoteze[1])
- optimizacijski postupak (vezan uz pretragu[2]) i
- funkcija cilja.

### 3.1.1. Model

Primjer jednostavne hipoteze $h_1 : \mathbb{R} \to \mathbb{R}$ koja, npr., opisuje ovisnost cijene stana $y$ o njegovoj površini $x$ jest:

$$h_1(x) = y = 2376.4x + 10250 \tag{3.1}$$

U slučaju da u obzir želimo uzeti i starost stana $x_2$ te udaljenost od centra grada $x_3$, imat ćemo nešto složeniju hipotezu $h_2 : \mathbb{R}^3 \to \mathbb{R}$ (3.2).

$$h_2(x) = y = 2792.6x_1 - 265.3x_2 - 354.1x_3 + 13250 \tag{3.2}$$

Navedene hipoteze su linearne hipoteze opće definicije $h : \mathbb{R}^n \to \mathbb{R}$. Skup hipoteza u koje spadaju $h_1$ i $h_2$ čine *linearni model* $H$. Općenito, linearni model je oblika

[1] Model definira skup mogućih hipoteza.

[2] Optimizacijskim postupkom pretražuje se prostor hipoteza.

$$h_{linear}(\mathbf{x}) = \sum_{i=1}^{N} w_i x_i + w_0 \tag{3.3a}$$

$$= \mathbf{w}^\top \mathbf{x} + w_0, \tag{3.3b}$$

gdje su $\mathbf{w}$ i $\mathbf{x}$ vektori[3]. Vektor $\mathbf{w}$ je skup koeficijenata koji se nalaze uz svaki od ulaznih elemenata. Koeficijenti se još nazivaju parametri modela ili *težine*. Skup svih vektora $\mathbf{w}$ određuje skup svih hipoteza danog modela.

U stvarnosti, većina veza ulaznih primjera $\mathbf{x^{(i)}}$ i izlaza $y^{(i)}$ nije linearna, stoga se ulazni primjeri preslikavaju u tzv. *prostor značajki*. Preslikavanje obavljaju nelinearne funkcije ulaznih varijabli koje se nazivaju bazne funkcije. Funkcija preslikavanja opisana je izrazom 3.4, a model s ugrađenom funkcijom preslikavanja je zadan s 3.5:

$$\boldsymbol{\phi} : \mathbb{R}^n \to \mathbb{R}^{m+1} : \boldsymbol{\phi}(\mathbf{x}) = (\phi_0(\mathbf{x}), ..., (\phi_m(\mathbf{x}))) \tag{3.4}$$

$$h(\mathbf{x}; \mathbf{w}) = \mathbf{w}\boldsymbol{\phi}(\mathbf{x}) \tag{3.5}$$

### 3.1.2. Funkcija cilja

Funkcija cilja algoritma strojnog učenja modelira dobrotu modela na skupu za učenje. Cilj je, iz skupa ulaznih primjera $\mathbf{x^{(i)}}$, na izlazu iz modela $H$ dobiti stvarne vrijednosti $y^{(i)}$, odnosno, *naučiti* model $H$. Na kraju postupka dobiva se optimalna hipoteza $h$. Skup $D = \{\mathbf{x^{(i)}}, y^{(i)}\}$ je skup primjera za učenje, a vrsta učenja u kojoj su poznate stvarne vrijednosti $y^{(i)}$, naziva se nadziranim učenjem. Nadzirano učenje u kojem su izlazne vrijednosti $y$ kontinuirane naziva se regresijom. Klasifikacija je vrsta nadziranog učenja u kojem su izlazne vrijednosti $y$ elementi konačnog skupa oznaka, npr. $y = \{0, 1, 2\}$. Učenje modela provodi se uz pretpostavku da između ulaznih i izlaznih podataka skupa za učenje $D$ postoji veza. Veza između ulaznih i izlaznih podataka opisana je nekom funkcijom $f$.

Neka je za učenje modela $H$ odabrana regresija. Opažena izlazna vrijednost je $h(\mathbf{x}^{(i)})$, izlaz hipoteze za dani ulazni vektor $\mathbf{x}^{(i)}$. Pogreška opažene izlazne vrijednosti u odnosu na očekivanu izlaznu vrijednost stvarne funkcije $f$ je šum $\epsilon_i$ (izraz 3.6). Šum se modelira kao normalno distribuirana slučajna varijabla $\epsilon \sim \mathcal{N}(0, \sigma^2)$. Regresijom nad skupom $D$ dobiva se hipoteza $h$ kao aproksimacija funkcije $f$ (izraz 3.6). Normalna razdioba definirana je izrazom 3.7. Vjerojatnost oznake za zadani primjer je

[3]Element $w_0$ naziva se pomak i nije vezan uz ulaze $x_i$.

dana s 3.8, a vjerojatnost da je cijeli skup primjera $\mathbf{X}$ označen oznakama $\mathbf{y}$ je dana s 3.9.

$$h(\mathbf{x}^{(i)}) = f(\mathbf{x}^{(i)}) + \epsilon_i \tag{3.6}$$

$$p(Y = y|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp\Big( - \frac{(y-\mu)^2}{2\sigma^2}\Big) \tag{3.7}$$

$$p(y|\mathbf{x}) = \mathcal{N}(f(\mathbf{x}), \sigma^2) \tag{3.8}$$

$$p(\mathbf{y}|\mathbf{X}) = \prod_{i=1}^{N} p(y^{(i)|\mathbf{x}^(i)}) \tag{3.9}$$

Cilj je maksimimizirati vjerojatnost da je cijeli skup primjera $\mathbf{X}$ ispravno označen (izraz 3.9). Drugim riječima, traži se model koji oznake čini najvjerojatnijima. Zbog matematičke jednostavnosti, radi se s logaritmom izglednosti, tj. negativnom log izglednošću težina $\mathbf{w}$:

$$\begin{aligned}
\ln p(\mathbf{y}|\mathbf{X}, \mathbf{w}) &= \ln \prod_{i=1}^{N} p(y^{(i)}|\mathbf{x}^{(i)}) = \ln \prod_{i=1}^{N} \mathcal{N}(f(\mathbf{x}^{(i)}), \sigma^2) \\
&= \ln \prod_{i=1}^{N} \mathcal{N}(h(\mathbf{x}^{(i)}; \mathbf{w}), \sigma^2) \\
&= \ln \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\Big( - \frac{(y^{(i)} - h(\mathbf{x}^{(i)}; \mathbf{w}))}{2\sigma^2}\Big) \\
&= \underbrace{-N \ln(\sqrt{2\pi}\sigma)}_{konst.} - \frac{1}{2\sigma^2} \sum_{i=1}^{N} (y^{(i)} - h(\mathbf{x}^{(i)}; \mathbf{w}))^2 \\
&\approx -\frac{1}{2} \sum_{i=1}^{N} (y^{(i)} - h(\mathbf{x}^{(i)}; \mathbf{w}))^2
\end{aligned} \tag{3.10}$$

Pogreška se dakle mjeri kao zbroj kvadratnih odstupanja predviđene vrijednosti $h(\mathbf{x})$ i stvarne vrijednosti $y$. U slučaju klasifikacije, na izlazu se dobiju vjerojatnosti da pojedini ulazni primjeri $x^{(i)}$ pripadaju oznaci, odnosno, klasi $c_k$, gdje je $k \in skup\_klasa$. Funkcija pogreške u slučaju klasifikacije također se dobiva preko negativne log izglednosti uz pretpostavku Multinoullijeve razdiobe [4] (izraz 3.11). Funkcija pogreške u slučaju klasifikacije, dana izrazom 3.11, zapravo je pogreška unakrsne entropije.

$$E(\mathbf{w}|\mathcal{D}) = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_k^{(i)} \ln h_k(\mathbf{x}^{(i)}; \mathbf{w}). \tag{3.11}$$

Pogreška unakrsne entropije je minimalna onda kad su vektori $y_k$ tzv. *one-hot* vektori ispravne klase $y_k^t$. Takav vektor ima sve vrijednosti jednake 0, osim na indeksu koji označava ispravnu klasu. Izraz 3.11 tad je jednak 0.

### 3.1.3. Optimizacijski postupak

Optimizacijski postupak pretražuje N-dimenzionalni prostor parametara modela $w_i$ s ciljem minimizacije odabrane funkcije pogreške proizvoljnog modela $H$:

$$h^* = argminE(h|\mathcal{D}). \tag{3.12}$$

## 3.2. Unaprijedna mreža kao model

Umjetne neuronske mreže primjer su prirodom inspiriranog modela strojnog učenja iz kojeg se tek nedavno razvilo čitavo područje dubokog učenja. Osnovni oblik umjetnih neuronskih mreža su unaprijedne mreže.

### 3.2.1. Perceptron

Iz regresijskog modela jednostavno se može dobiti binarni klasifikacijski model. Osnovna je razlika klasifikacije u odnosu na regresiju ta što regresija na izlazu daje kontinuirane vrijednosti, a klasifikacija kategoričku vrijednost - klasu ulaznog primjera. Regresijski je model npr. linearan,

$$h_{reg}(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + w_0. \tag{3.13}$$

Ukoliko je izlazna vrijednost viša od $y_{granica}$, primjer ne pripada klasi $0$, nego klasi $1$. Funkcija $h_{reg}$ za potrebe ove transformacije postaje

$$h_{klas} = \begin{cases} 0, & \text{ako } \mathbf{w}^\top \mathbf{x} + w_0 \geq y_{granica} \\ 1, & \text{inače.} \end{cases} \tag{3.14}$$

Slična se transformacija može napraviti i s drugim nelinearnim funkcijama, npr. funkcijom praga (izraz 3.15) i sigmoidalnom funkcijom $h_{sigmoid}$ (izraz 3.15, slika 3.2a). Sigmoidalna funkcija koristi se i kao model logističke regresije (3.11). Modeli nastali ovom transformacijom nazivaju se poopćeni linearni modeli, a funkcija

koja obavlja transformaciju je *aktivacijska funkcija*. Može se reći da je izlaz poopćenog linearnog modela kompozicija aktivacijske funkcije i linearne regresije s obzirom na ulaze $\mathbf{x}$. Specifično, model koji koristi funkciju praga (3.16) naziva se *perceptron*.

$$h_{sigmoid}(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x}) \tag{3.15a}$$

$$= \frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{x})} \tag{3.15b}$$

$$h_{prag} = \begin{cases} +1, & \text{ako } \alpha \geq 0 \\ -1, & \text{inače.} \end{cases} \tag{3.16}$$

Funkcija je pogreške perceptrona dana izrazom 3.17 i kažnjava samo negativno klasificirane primjere. U slučaju da je $y^{(i)}$ jednak $+1$, a izraz $-\mathbf{w}^\top \mathbf{x}^{(i)} y^{(i)}$ pozitivan, umnožak je pozitivan pa je čitav izraz negativan, dakle, kazna za taj primjer je $0$. Isto se događa i u slučaja da su obje komponente negativne.

$$E(\mathbf{w}|\mathcal{D}) = -\sum_{i=1}^{N} max(0, -\mathbf{w}^\top \mathbf{x}^{(i)} y^{(i)}). \tag{3.17}$$

Za danu funkciju pogreške perceptrona 3.17 ne postoji rješenje u zatvorenoj formi kao kod funkcije pogreške linearne regresije (3.12), stoga treba posegnuti za iterativnim optimizacijskim postupkom. Iterativni optimizacijski algoritam perceptrona je *gradijentni spust* i temeljni je algoritam dubokog učenja.

### 3.2.2. Gradijentni spust

Gradijentni spust jednostavan je iterativni postupak optimizacije - pretraživanja prostora parametara modela $\mathbf{w}$ s ciljem pronalaske optimalne hipoteze $h$. Koristi se za učenje modela perceptrona, ali i svih algoritama vezanih uz klasične neuronske i duboke mreže.

Neka je funkcija pogreške $f_{prag}$ (3.16) za slučaj perceptrona. Vektor gradijenta funkcije kazuje u kojem smjeru vrijednost funkcijen najbrže raste (3.18). Minimum se, dakle, nalazi krećući se u smjeru suprotnom od gradijenta (3.19).

$$\nabla f = (\frac{\partial f(\mathbf{x})}{\partial x_0}, \frac{\partial f(\mathbf{x})}{\partial x_1}, ..., \frac{\partial f(\mathbf{x})}{\partial x_n}). \tag{3.18}$$

$$\mathbf{x} \leftarrow \mathbf{x} - \eta \nabla E(\mathbf{w}|\mathcal{D}) \tag{3.19}$$

Gradijent funkcije gubitka za netočno klasificirane primjere dobiva se parcijalno derivirajući funkciju praga $f_{prag}$ po parametrima modela $\mathbf{w}$ (3.20). Postupak učenja perceptrona svodi se na *ažuriranje težina* po Windrow-Hoffovo pravilu [5] danom izrazom 3.20. Koeficijent koji definira veličinu "kazne" modela $\eta$ je stopa učenja i jedan je od najbitnijih hiperparametara modela strojnog učenja.

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E(\mathbf{w}|\mathcal{D}). \tag{3.20}$$

Algoritam perceptrona iterativni je algoritam pa ga je prigodno prikazati u obliku pseudokoda:

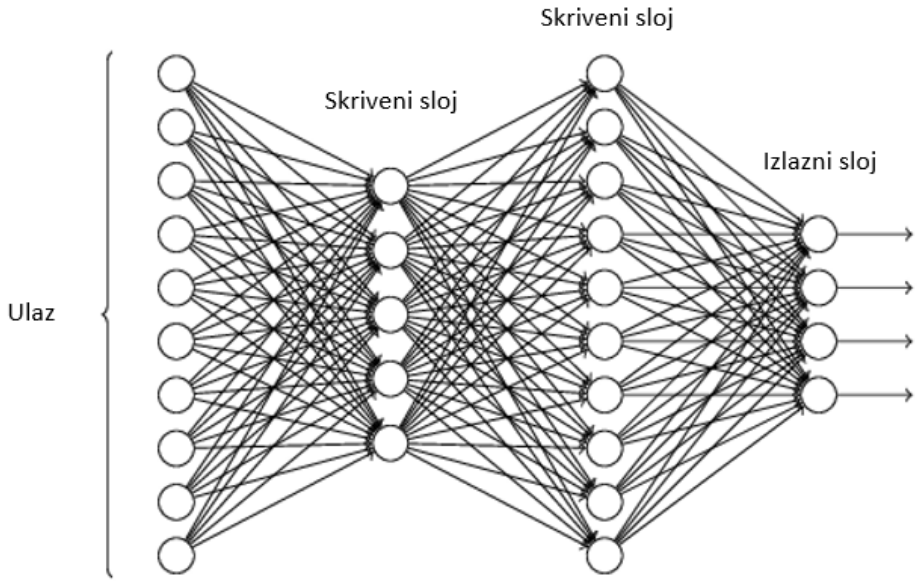
**Algoritam 1:** Algoritam perceptrona

**Funkcija** `NadiHipotezu()`**:**
- Inicijaliziraj $\mathbf{w} \leftarrow (0, ..., 0)$
- **dok** *nije zadovoljen uvjet konvergencije* **čini**
  - **za** *i = 1, ..., N* **čini**
    - **ako** $f(\boldsymbol{w}^\top \boldsymbol{x}^{(i)}) \neq y^{(i)}$ **onda**
      - $\mathbf{w} \leftarrow \mathbf{w} + \eta \mathbf{x}^{(i)} y^{(i)}$
    - **kraj**
  - **kraj**
- **kraj**
- **vrati w**

Algoritam kreće s parametrima modela postavljenima na $0$. Težine je moguće postaviti i slučajno unutar nekog intervala, npr. $[-1, 1]$. Uvjet konvergencije, u ovom slučaju, ispituje se tek nakon prolaska kroz cijeli skup podataka. To može biti provjera funkcije pogreške kojom ispitujemo je li pogreška modela pala ispod neke odabrane vrijednosti. Ako jest, funkcija vraća naučeni skup parametara koji definira traženu hipotezu. U suprotnom, postupak ponovno prolazi kroz skup podataka. Nakon svakog ulaznog primjera, ažuriraju se vrijednosti parametara $w$. Učenje koje nakon svakog primjera ažurira parametre modela naziva se *online* učenje.

### 3.2.3. Unaprijedna mreža

Unaprijedna mreža sastoji se od posebno uređene mreže poopćenih linearnih modela koji se nazivaju jedinice. Najčešća aktivacijska funkcija jedinica jest zglobnica [6] opisana izrazom 3.24 i slikom 3.2b). Svaka jedinica ima svoje ulaze i izlaze, a njihov broj ovisi o arhitekturi mreže. Jedinice prvog sloja na ulaze primaju ulaze modela $\mathbf{x}$. Točno kao i kod linearne regresije, svaki ulaz $x_i$ množi se s pripadnom težinom




**Slika 3.1:** Primjer unaprijedne mreže. Jedinice su prikazane krugovima. Strelice označavaju smjer ulaznih podataka prema izlazu. Mreža ima 4 izlaza (svaki od izlaza može predstavljati jednu klasu u klasifikacijskom problemu) i 2 skrivena sloja sa po 5 i 10 jedinica u svakom. Izvor: [34]

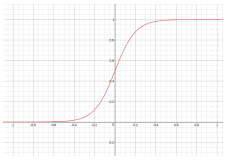
$w_{i1}$, gdje drugi indeks označava sloj mreže. Taj se rezultat "provlači" kroz pripadnu aktivacijsku funkciju i dalje prosljeđuje unaprijednim slojevima sve do izlaza. Primjer unaprijedne mreže dan je na slici 3.1.

$$f_{ReLU} = max(0, x). \tag{3.21}$$

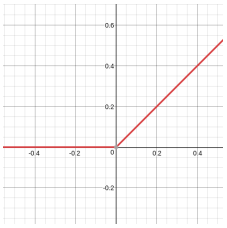
Informacija kroz mrežu uvijek struji od ulaza prema izlazu. Unaprijedna mreža nema petlji. Svaka jedinica obavlja kompoziciju aktivacijske funkcije i zbroja umnožaka ulaza i parametara. Mrežu onda možemo prikazati kao niz kompozicija jednostavnijih funkcija (3.22):

$$f(\mathbf{x}, \mathbf{w}) = o(f_L(f_{L-1}(...(f_1(\mathbf{x}, \mathbf{w_1})), ...), w_{L-1}), w_L), \tag{3.22}$$

Može se reći da model odgovara usmjerenom acikličkom grafu koji kazuje kako su funkcije $(f_L, f_{L-1}, ..., f_1)$ povezane. Izraz 3.20 postaje nečitljiv za dublje mreže pa se model izražava preko pomoćnih varijabli $h_L, h_{L-1}, ..., h_1$ kao (3.23):



**(a)** Sigmoidalna funkcija



**(b)** Zglobnica

**Slika 3.2:** Sigmoidalna funkcija (a) i zglobnica (b).

$$\mathbf{h}_1 = f_1(\mathbf{x}, \mathbf{w_1}) \tag{3.23a}$$

$$\vdots$$

$$\mathbf{h}_{L-1} = f_{L-1}(\mathbf{h}_{L-2}, \mathbf{w}_{L-1}) \tag{3.23b}$$

$$\mathbf{h}_L = f_L(\mathbf{h}_{L-1}, \mathbf{w}_L) \tag{3.23c}$$

$$f(\mathbf{x}, \mathbf{w}) = o(\mathbf{h}_L). \tag{3.23d}$$

Sad je jasnije vidljivo kako broj $L$ označava broj slojeva mreže i može se reći da je i on, uz stopu učenja, hiperparametar modela. Gledajući korak dalje, jedino su ulaz $\mathbf{X}$ i izlaz $\mathbf{y}$ specificirani, a skriveni slojevi biraju se na način koji osigurava najbolju aproksimaciju funkcije.

Ideja unaprijednog modela je ostvariti linearnu kombinaciju nelinearnih funkcija. Nelinearnost se ostvaruje nelinearnim aktivacijskim funkcijama jedinica, npr. zglobnicom. Zglobnica je pogodna jer je računski nezahtjevna, a dijelom riješava problem iščezavajućeg gradijenta [37].

Valja primijetiti kako unaprijedni model uči izravno iz skupa podataka $\mathbf{X}$. Učenje parametara linearnog modela $(\mathbf{w}, b)$ spregnuto je, dakle, s učenjem parametara $\mathbf{\Phi}_\phi$ koji određuju reprezentaciju ulaznog vektora $\mathbf{x}$. Cijena je ove fleksibilnosti ta da optimizacijski problem više nije konveksan [24]. Ukupan skup parametara je $\mathbf{\Phi} = (\mathbf{w}, b) \cup \mathbf{\Phi}_\phi$, a klasa funkcija $\mathbf{\Phi}_\phi$ određena je strukturom mreže.

### 3.2.4. Algoritam unazadne propagacije

Učenje mreže provodi se ažuriranjem parametara $w_{ij}$ unaprijednog modela ovisno o odabranoj funkciji pogreške. Svaka je jedinica u određenoj je mjeri odgovorna za izlaz mreže. Jedinice posljednjeg skrivenog sloja neposredno su odgovorne za izlaz. Za njih se računaju parcijalne derivacije u obliku:

$$\frac{\partial l_{CE}(y_t, o(\mathbf{h}_L))}{\partial \mathbf{w}^L} = \frac{\partial l_{CE}(y_t, \hat{y})}{\partial \mathbf{w}^L}, \tag{3.24}$$

gdje je $l_{CE}$ po dijelovima glatka funkcija pogreške modela, $\hat{y}$ oznaka izlaza modela, a $y_t$ oznaka očekivanog izlaza. Parcijalne derivacije funkcije $l_{CE}$ po parametrima posljednjeg skrivenog sloja $\mathbf{w}^L$ daju gradijente za ažuriranje pripadnih parametara. Dobiveni gradijenti propagiraju se unatrag kroz mrežu $(L-1)$-om skrivenom sloju sve do prvog skrivenog sloja (3.26). Pritom se koristi pravilo ulančavanja opisano s:

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} = \frac{df(y)}{dy}\frac{dg(x)}{dx}, y = g(x), z = f(y) = f(g(x)). \tag{3.25}$$

$$\frac{\partial l_{CE}(y_t, o(\mathbf{h}_L))}{\partial \mathbf{w^l}} = \frac{\partial l_{CE}(y_t, \hat{y})}{\partial \mathbf{w^l}} \tag{3.26a}$$

$$\frac{\partial l_{CE}(y_t, \hat{y})}{\partial \mathbf{w^l}} = \frac{\partial l_{CE}(y_t, \hat{y})}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial \mathbf{h}^L}\frac{\partial \mathbf{h}^L}{\partial \mathbf{h}^{L-1}}...\frac{\partial \mathbf{h}^{l+1}}{\partial \mathbf{h}^l} \tag{3.26b}$$

$$\frac{\partial l_{CE}(y_t, \hat{y})}{\partial \mathbf{w^l}} = \frac{\partial l_{CE}(y_t, \hat{y})}{\partial \hat{y}}\frac{\partial o(\mathbf{h}^L)}{\partial \mathbf{h}^L}\frac{\partial f^L(\mathbf{h}^{L-1}, \mathbf{w}^L)}{\partial \mathbf{h}^{L-1}}...\frac{\partial f^l(\mathbf{h}^{l-1}, \mathbf{w}^l)}{\partial \mathbf{h}^l} \tag{3.26c}$$

## 3.3. Konvolucijska mreža kao model

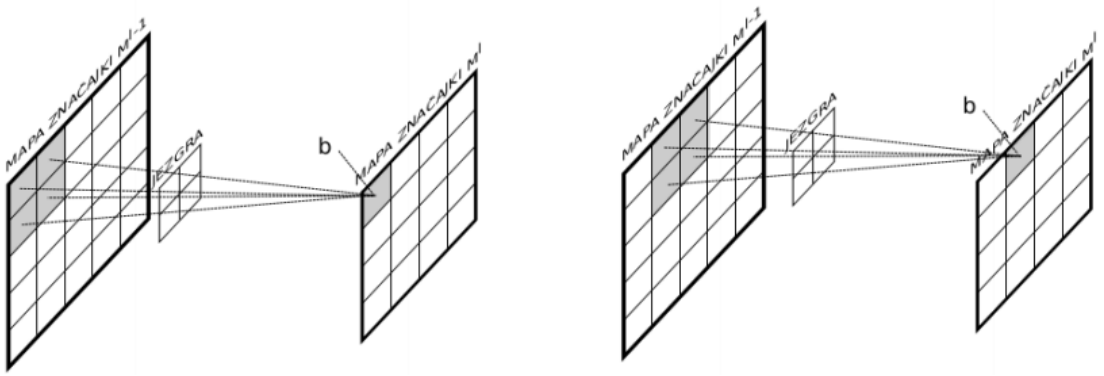
Razvoj GPU-ova potaknuo je ponovni interes za unaprijednim mrežama jer su GPU-ovi zajedno s ostalim hardverom omogućili neusporedivo brži račun od računala s kraja 80-ih. Osim opisanih unaprijednih modela, u igru su došli i drugi modeli iz iste porodice koji su predloženi u to doba, ali su ostali samo na razini zanimljivog koncepta. Tu su prije svega konvolucijski modeli. Oni su, zbog načina na koji uče, postali temeljna komponenta u problemima računalnog vida i najjednostavnije ih je uvesti u kontekstu slika.

### 3.3.1. Konvolucijski sloj

Temeljna operacija konvolucijskog sloja je *konvolucija*. U slučaju dvodimenzionalnog ulaza (npr. slike), definirana je kao:

$$S(i,j) = (K * I)(i,j) = \sum_{m=m_{min}}^{m_{max}} \sum_{n=n_{min}}^{n_{max}} I(i+m, j+n)K(m,n). \tag{3.27}$$

Indeksi $i$ i $j$ označavaju piksele slike $I$. $K$ označava jezgru ili filtar, matricu dimenzija manjih od dimenzije slike koja predstavlja "prozor" kojim prolazimo kroz




**Slika 3.3:** Operator konvolucije na primjeru mape značajki kvadratnih dimenzija (5x5) i jezgre dimenzije 2x2. Jezgra je "prozor" kojim prolazimo kroz ulaznu mapu značajki sloja i na izlazu dobivamo novu mapu, manjih dimenzija. Izvor: [24].

dvodimenzionalni ulaz (slika 3.3). Rasponi *min* i *max* određeni su vrijednostima na kojima su $I$ i $K$ definirani (tj. $\neq 0$). Konvolucijom se dobiva skalarna vrijednost za svaki par indeksa $(i, j)$ - time se smanjuju dimenzije izlazne slike u odnosu na ulaznu. Sloj unaprijednog modela koji koristi konvoluciju naziva se konvolucijski sloj. Ulazna i izlazna slika konvolucijskog sloja općenito je *mapa značajki*. U slučaju slike u boji (RGB), tri su ulazne mape značajki, a broj izlaznih mapa značajki je proizvoljan. S $C$ se označava broj ulaznih mapa značajki u 3.28. Za svaku izlaznu mapu značajki koristi se zasebna jezgra.

$$S(c,i,j) = (K * I_c)(c,i,j) = \sum_{c \in C} \sum_{m=m_{min}}^{m_{max}} \sum_{n=n_{min}}^{n_{max}} I(m,n,c) K(i-m, j-n, c). \quad (3.28)$$

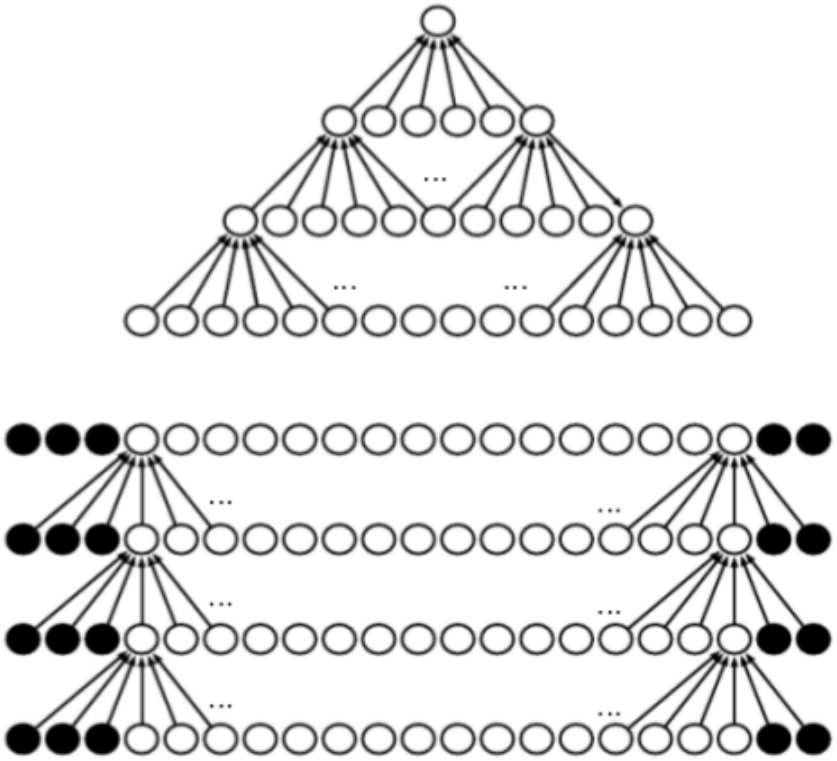
Konvolucijski sloj ne smanjuje nužno veličinu izlaznih mapa značajki u odnosu na ulazne. To se ostvaruje nadopunom (slika 3.4).

Jezgra $k$-te mape značajki je zajednička svim jedinicama te mape značajki, za razliku od jedinica prethodno opisanog unaprijednog modela, gdje je svaka jedinica trenutnog sloja povezana sa svakom jedinicom sljedećeg sloja (slika 3.5a). Dvije su bitne posljedice ovog svojstva konvolucijskog sloja:

- lokalnost značajki (slika 3.5b) i

- puno manji broj parametara u odnosu na klasični unaprijedni model.

Lokalnost značajki intuitivno je korisna pri analizi informacije slike. Konvolucijski sloj, prolazeći jezgrom kroz danu mapu značajki, traži karakteristične detalje i dijelove i, naišavši na svojstvo koje traži, na izlazu daje veću vrijednost, tj. "okida" na dani ulaz. To upravo odgovara ulozi jezgre! Upravo je poželjno ograničiti kapacitet jezgre na jedno specifično svojstvo. Početni konvolucijski slojevi karakteristični su po tome



**Slika 3.4:** Nadopunjavanje u slučaju 1D konvolucije. Izvor: [24].

da okidaju na rubove objekata na slici. Razlog je to što je njihovo *receptivno polje* relativno malo. Jezgre su obično veličine od 3x3 do 7x7. Za veličinu slike 256x412, jezgra uspije obuhvatiti tek male dijelove slike.

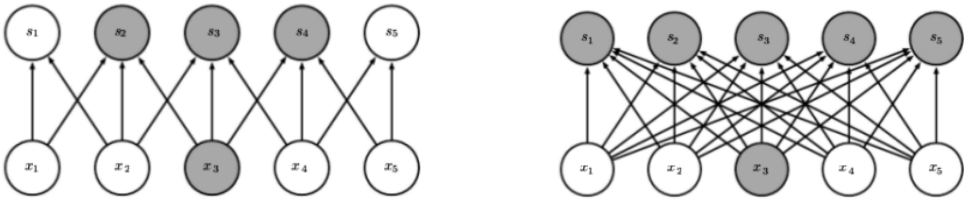
Receptivno polje kasnijih konvolucijskih slojeva je veće. To svojstvo opisano je na slici 3.6. Veće receptivno polje omogućava jedinicama kasnijih konvolucijskih slojeva sažimanje veće količine informacija. Jezgre kasnijih slojeva imaju kapacitet prepoznavanja apstraktnijih informacija o ulazu, npr. očiju ili lica na primjeru slika.

Druga posljedica lokalne povezanosti jedinica jest puno manji broj parametara konvolucijskog modela, što konvolucijski model čini računski znatno manje zahtjevnim.

### 3.3.2. Primjeri konvolucijskih arhitektura

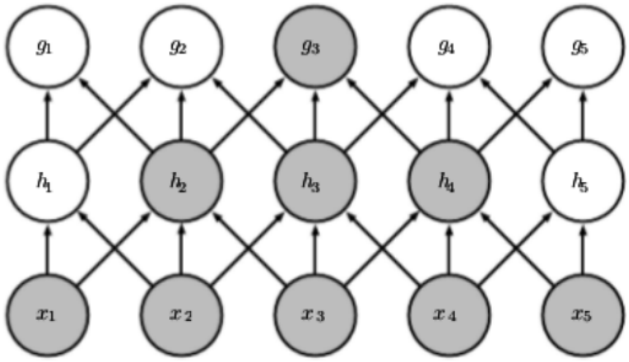
Koristeći konvolucijski model za klasifikaciju, izlaz mora biti vektor vjerojatnosti pojedine oznake $y_i$ - iz slike moramo dobiti vektor. Da bi se smanjila dimenzionalnost mapa značajki nekog sloja, nad tim slojem provodi se konvolucija (bez nadopunjavanja). Time se povećava broj parametara modela. Dakle, broj parametara je usko vezan uz veličinu ulazne slike.

Ipak, postoje načini kako iz velikog ulaznog tenzora dobiti vektor na izlazu iz mo-

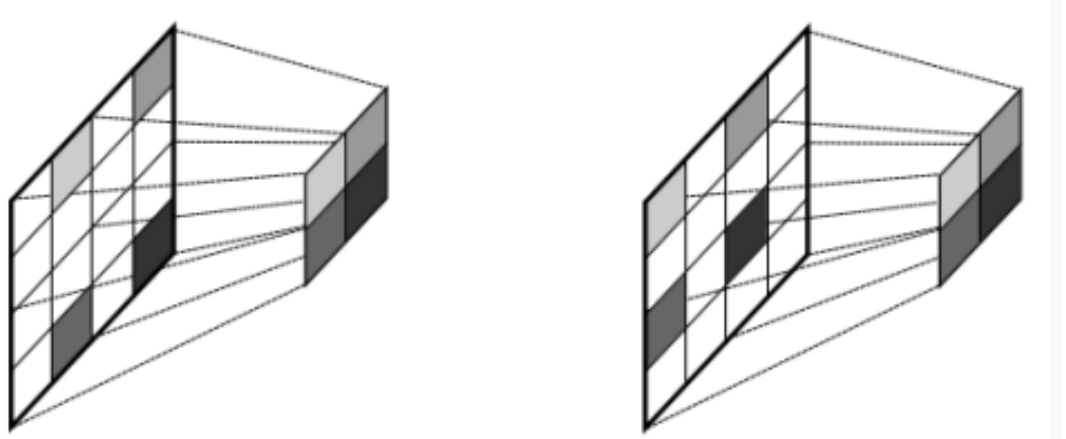



**(a)** Konvolucijski sloj. **(b)** Potpuno povezani sloj.

**Slika 3.5:** Razlika konvolucijskog sloja i potpuno povezanog sloja na primjeru jednodimenzionalnog ulaza. Jedna dimenzija u primjerima korištena je zbog jednostavnosti prikaza. Izvor: [24].




**Slika 3.6:** Povećanje receptivnog polja kroz konvolucijski model. Jezgra sažima svake tri vrijednosti trenutnog sloja u jednu vrijednost sljedećeg sloja. U trećem retku prikazano je pet ulaza koji se sažimaju u tri, nakon čega se u prvom retku ta tri sažimaju u jedan. Ta jedinica ($g_3$) sažima informaciju početnih pet jedinica ($x_1$ do $x_5$). Kaže se da je receptivno polje jedinice $g_3$ jednako pet. Primjer opisuje konvoluciju nad jednodimenzionalnim ulazom zbog jednostavnosti prikaza. Izvor: [24].

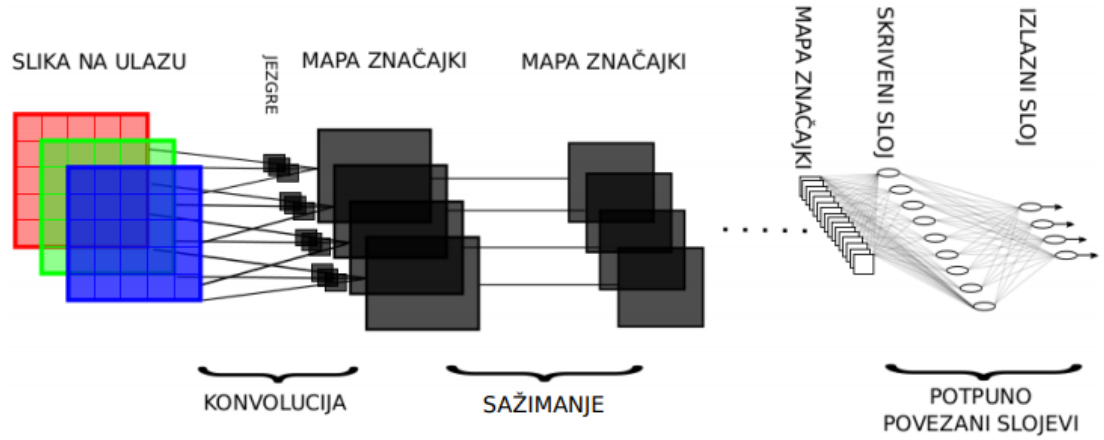


**Slika 3.7:** Sloj sažimanja. Operator sloja je neka agregacijska funkcija, npr. maksimum ili prosječna vrijednost. Slojem sažimanja smanjuje se dimenzija mapa značajki bez uvođenja novih parametara. Izvor: [24].

dela bez "potrošnje" velikog broja parametara. *Sloj sažimanja* također koristi prozore za prolazak kroz ulazne mape značajki. Operator sloja sažimanja je agregacijska funkcija, npr. $max$ - iz svakog prozora uzima se maksimalna vrijednost (slika 3.7). Još jedan sloj koji je karakterističan za konvolucijski klasifikator je potpuno povezani sloj klasičnih neuronskih mreža. Kad se mape značajki svedu na "piksele značajki" (dimenzija svake mape bude 1x1), za posljednji sloj obično se postavlja potpuno povezani sloj koji transformira vektor mapa u izlazni vektor čija veličina odgovara broju oznaka klasifikacijskog zadatka (slika 3.8).

Čitav konvolucijski model prikazan je na slici 3.5. Na ulazu je slika s tri ulazna kanala. Ti kanali prolaze kroz prvi konvolucijski sloj. U njemu se dane četiri jezgre pa su na izlazu četiri mape značajki dimenzija manjih ili jednakih dimenzijama ulaznih kanala (ovisno o tome koristi li se nadopuna ili ne). Slijedi niz konvolucijskih slojeva i slojeva sažimanja kojima se dodatno smanjuju dimenzije mapa značajki. Konačno, mape značajki "spljoštene" su u vektor (N mapa značajki dimenzije 1x1). Na taj vektor primijeni se potpuno povezani sloj kojim se vektor dimenzija N pretvara u vektor dimezija $C$ - broja oznaka.

Primjer konkretnog konvolucijskog modela je VGG16. On na ulazu prima RGB sliku veličine 224x224. Broj ulaznih mapa značajki prvog sloja modela je 3, a broj izlaznih je 64. To povlači da VGG u svom prvom sloju ima 64 jezgre koje obavljaju konvoluciju po trima ulaznim mapama značajki. Nakon još jednog konvolucijskog sloja, slijedi sloj sažimanja koji dvostruko smanjuje dimenziju širine i visine "slike", odnosno, mapa značajki na 112x112. Nakon toga slijede slojevi konvolucije koji pove-




**Slika 3.8:** Konvolucijski model. Izvor: [24].

ćavaju broj izlaznih značajki na 128. Nakon niza slojeva sažimanja i slojeva konvolucije, ulaz je pretvoren u tenzor dimenzija 7x7x512. Nakon toga slijedi potpuno povezani sloj koji uzima svaku vrijednost tog tenzora i povezuje ju s tenzorom 1x1x4096, odnosno vektorom. Ulaz je potpuno "izravnan". Slijedi nekoliko potpuno povezanih slojeva koji u konačnici daju vektor od Y vrijednosti koje predstavljaju vjerojatnosti za svaku od Y oznaka.

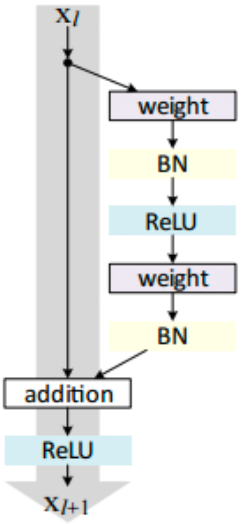
VGG16 primjer je modela koji je vrlo zahtjevan za učenje i to upravo zbog potpuno povezanih slojeva koji znatno povećavaju broj parametara, a karakteristika je konvolucijskih arhitektura da taj broj smanjuju. Drugi je problem kod učenja ovog modela taj što je model relativno dubok (19 slojeva). Informacija koja se unazadno propagira svakim slojem postaje sve beznačajnija (zbog ulančanog množenja). Drugi je problem riješen uvođenjem *preskočnih veza* koji su karakteristički za ResNet arhitekturu.

Primjer ResNet modela je ResNet50 [7]. Preskočna veza opisana je slikom 3.9. Izlaz sloja dijeli se u dvije grane. Jedna grana prolazi redom kroz slojeve, dok je druga grana direktno vezana na neki od budućih slojeva. Postiže se to da grana koja preskače slojeve čini manje duboki model pa se informacija lakše propagira unatrag. ResNet arhitektura omogućila je učenje znatno dubljih modela (150-200 slojeva).

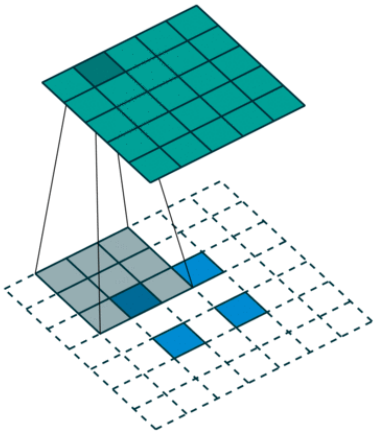
### 3.3.3. Enkoder-dekoder arhitektura

Receptivno je polje kasnijih slojeva dubokog modela nužno veće od receptivnog polja ranijih slojeva. Na primjeru modela koji se sastoji samo od $L$ konvolucijskih slojeva i jezgara veličine $KxK$, receptivno je polje posljednjeg sloja veličine $L * (K - 1) + K$.

Umjesto sažimanja ulaza do veličine izlaznog vektora, moguće ga je sažeti do određenog, eksperimentalno određenog oblika, a onda takvu sažetu, jezgrovitu informaciju




**Slika 3.9:** Preskočna veza. Izvor: [36]



**Slika 3.10:** Dekonvolucija. Izvor: [33].

bolje iskoristiti. Konkretno, dimenzije ulaznih mapa moguće je i povećati, ne samo smanjiti. Postoji nekoliko načina na koje se to može napraviti, a primjer je dekonvolucija (slika 3.10) [33]. Pravilno je reći da je dekonvolucija zapravo obrnuta konvolucija, a način na koji se računaju konkretne izlazne vrijednosti ovisi o implementaciji.

Izlaz modela više neće biti vektor vjerojatnosti. Izlaz modela sad je tenzor željenih dimenzija koje podešavamo brojem i svojstvima dekonvolucija. Čitav postupak povećanja dimenzija ulaznih mapa značajki naziva se naduzorkovanje (eng. *upsampling*). Model se može podijeliti na dio prije središnjeg sloja (*enkoder*) i dio nakon središnjeg sloja (*dekoder*). Duboki model u poglavlju 4 koristi enkoder-dekoder arhitekturu (slika 4.4).

# 4. Nenadzirano učenje procjene geometrijskih značajki

## 4.1. Ideja i komponente modela

U nastavku rada opisan je model dubokog učenja za zajedničku procjenu dubine i vlastitog gibanja nenadziranim učenjem. Prije opisa modela, valja istaknuti nekoliko činjenica. Dubinske mape koje se procjenjuju govore o relativnim udaljenostima između točaka scene, odnosno, ne procjenjuju se apsolutne udaljenosti od promatrača. Tijekom učenja nisu poznati ni dubina ni vlastito gibanje. Dostupne su jedino ulazne slike koje se slažu u nizove[1]. Model dubokog učenja sastoji se od dvije građevne jedinice - jedinice za procjenu dubine i jedinice za procjenu vlastitog gibanja. Arhitektura građevnih jedinica opisana je u ovom poglavlju.

Poglavlje je podijeljeno u dva dijela. U drugom dijelu opisana je iterativna metoda najbližih točaka i način na koji se ta metoda koristi kao komponenta funkcije gubitka. U prvom dijelu opisani su svi ostali dijelovi dubokog modela. Duboki model ima složenu funkciju gubitka. Funkcija gubitka modela zbroj je četiri komponente:

- reprojekcijske pogreške,
- pogreške glatkosti dubinske mape,
- pogreške strukturne sličnost i
- pogreške iterativne metode najbližih točaka.

Opisana je i formalno definirana uloga svake komponente funkcije gubitka.

### 4.1.1. Reprojekcija slike

Neka je oblak točaka $Q_t^{ij}$ dan jednakošću 4.1. Matrica $D_t^{ij}$ procjena je dubinske mape u trenutku $t$, $K$ je intrinzična matrica, a $[i, j, 1]$ je piksel na poziciji $(i, j)$ izražen homogenim koordinatama (kao u 2.18). Transformacijska matrica $T_{t\to t+1}$ procjena je

[1] Način na koji se slike slažu u nizove opisan je u poglavlju 5.

vlastitog gibanja iz trenutka $t$ u trenutak $t+1$. Ona se, zbog jednostavnosti, obilježava s $T_{t+1}$. Množeći oblak točaka $Q_t^{ij}$ slijeva matricom vlastitog gibanja dobiva se projekcija oblaka točaka $\widetilde{Q}_{t+1}^{ij}$ (4.2) u budućem trenutku. Konačno, oblak točaka projicira se natrag na slikovno platno množenjem slijeva intrinzičnom matricom K (4.3).

Izrazom 4.3 definirana je projekcija piksela $p_t = (i, j)$ u piksel $\widetilde{p}_{t+1} = (\widetilde{i}, \widetilde{j})$. Projekcija se označi s $\widetilde{X}_{t+1}^{ij}$. Projekcija piksela radi se, osim u smjeru $t \to t+1$, i u obrnutom smjeru, $t \to t-1$ (4.4) - iz trenutnog u prošli vremenski trenutak. Simbol $\sim$ iznad elemenata u izrazima 4.1, 4.2, 4.3 i 4.4 označava projekcije, a procjene su označene pojačanim tekstom.

$$Q_t^{ij} = \mathbf{D_t^{ij}} K^{-1}[i, j, 1]^\top \tag{4.1}$$

$$\widetilde{Q}_{t+1}^{ij} = \mathbf{T_{t+1}} \mathbf{D_t^{ij}} K^{-1}[i, j, 1]^\top \tag{4.2}$$

$$\widetilde{X}_{t+1}^{ij} = X_t^{\widetilde{ij}}[\widetilde{i}, \widetilde{j}, 1]^\top = K\mathbf{T_{t+1}}(\mathbf{D_t^{ij}} K^{-1}[i, j, 1]^\top) \tag{4.3}$$

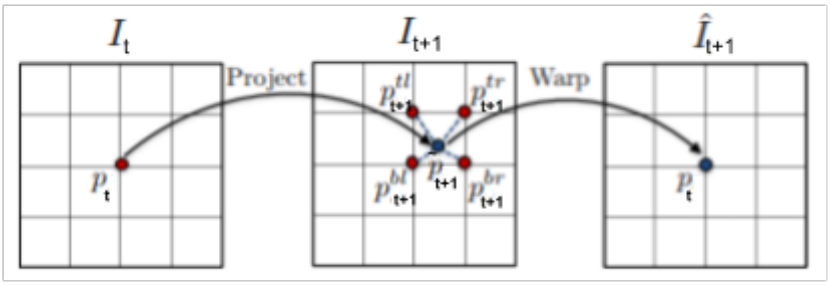
$$\widetilde{X}_{t-1}^{ij} = X_t^{\widetilde{ij}}[\widetilde{i}, \widetilde{j}, 1]^\top = K\mathbf{T_{t-1}}(\mathbf{D_t^{ij}} K^{-1}[i, j, 1]^\top) \tag{4.4}$$

Piksel $p_t = [i \in \mathbb{N}, j \in \mathbb{N}, 1]$ slike $I_t$ projicira se u piksel $\widetilde{p}_{t+1} = [\widetilde{i} \in \mathbb{R}, \widetilde{j} \in \mathbb{R}, 1]$ slike $I_{t+1}$. Koordinate projiciranog piksela $\widetilde{p}_{t+1}$ su realne pa je potrebno napraviti linearnu interpolaciju najbližih susjednih piksela kako bi se odredila njegova vrijednost. Težinski se zbrajaju vrijednosti najbliža četiri piksela ovisno o udaljenosti projiciranog piksela $\widetilde{p}_{t+1}$ do svakog susjednog. Sad kad je određena vrijednost projiciranom pikselu $\widetilde{p}_{t+1}$, on se *reprojicira* natrag na originalnu koordinatu $p_t$ (slika 4.1). Postupak se ponavlja za svaki piksel trenutne slike $I_t$, a rezultat postupka je reprojekcijska slika $\hat{I}_{t+1}$ (slika 4.1, izraz 4.5). U slučaju reprojekcije iz sadašnjeg trenutka u prošli, dobiva se reprojekcijska slika $\hat{I}_{t-1}$.

$$\hat{I}_{t+1}(i, j) = I_{t+1}(\hat{i}, \hat{j}) = \sum_{x \in \{g,d\}, y \in \{l,d\}} w^{xy} I_{t+1}(x, y) \tag{4.5}$$

### 4.1.2. Maska preklapanja

U kontekstu projekcije slike iz sadašnjeg u prethodni trenutak, valja pojasniti pojam maske preklapanja. U slučaju gibanja prema naprijed, na primjer, slika $I_t$ u trenutku $t$ približena je u odnosu na sliku $I_{t-1}$ u trenutku $t-1$. Dio scene vidljiv sa slike $I_t$

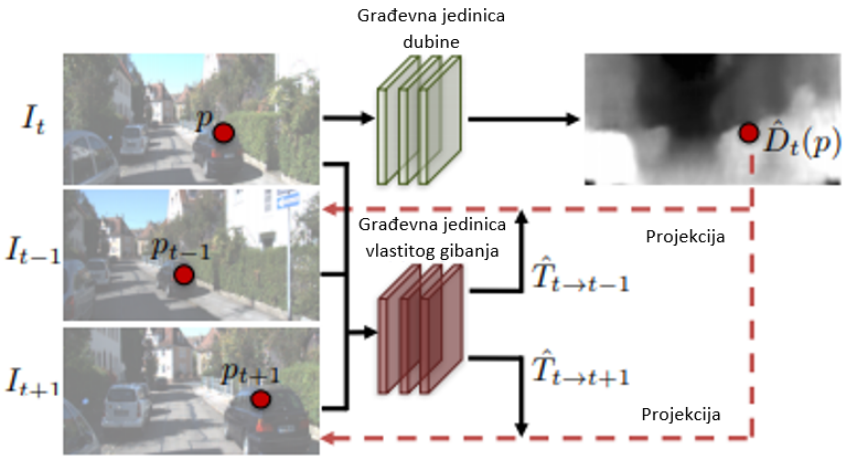



**Slika 4.1:** Projekcija i reprojekcija piksela $p_t$ u vremenskom trenutku $t$ u budući vremenski $t+1$. Izvor: [10].

**Slika 4.2:** Maska preklapanja.

vidljiv je i sa slike $I_{t-1}$, ali ne vrijedi obrnuto. Projekcijom slike $I_t$ u prošli trenutak dogodit će se, stoga, da će neki pikseli biti projicirani izvan okvira slike. Ti pikseli ignoriraju se maskom preklapanja $M^{ij}$ definiranoj u 4.6. Primjer maske preklapanja dan je na slici 4.2. U prvom redu dana je slika u trenutku $t$. U drugom redu dana je slika u trenutku $t-1$. U trećem redu dana je reprojekcija $\widetilde{I}_{t-1}$. Maska preklapanja dana je u četvrtom redu.

$$M(i,j) = \begin{cases} 0, & \text{ako } (\widetilde{i} < 0) \bigcup (\widetilde{i} > H) \bigcup (\widetilde{j} < 0) \bigcup (\widetilde{j} > W) \\ 1, & \text{inače.} \end{cases} \tag{4.6}$$




**Slika 4.3:** Građevne jedinice dubokog modela. Izvor: [10].

### 4.1.3. Od ulaza do funkcije pogreške

Model, u svakoj iteraciji, na ulazu prima niz kronološki poredanih slika (..., $I_{t-2}$, $I_{t-1}$, $I_t$, $I_{t+1}$, $I_{t+2}$, ...)[2]. Ključ je iskoristiti vezu između središnje slike i svake od ostalih pod pretpostavkom pomaka između slika. Nad nizom slika, procjenjuje se dubina središnje slike $D_t$ i vlastita gibanja između središnje i svake od ostalih slika. Dobivaju se reprojekcije slika (..., $\hat{I}_{t-2}$, $\hat{I}_{t-1}$, $\hat{I}_{t+1}$, $\hat{I}_{t+2}$, ...). Reprojekcijske slike $\hat{I}$ uspoređuju se s originalnim slikama $I$. Razlike u vrijednostima onih piksela koji nisu maskirani maskom preklapanja (4.6) određuju reprojekcijsku pogrešku. Uzima se srednja vrijednost zbroja razlika svih nemaskiranih piksela (4.7). Izraz, zbog jednostavnosti, prikazuje samo pogrešku reprojekcije za sliku $\hat{I}_{t+1}$.

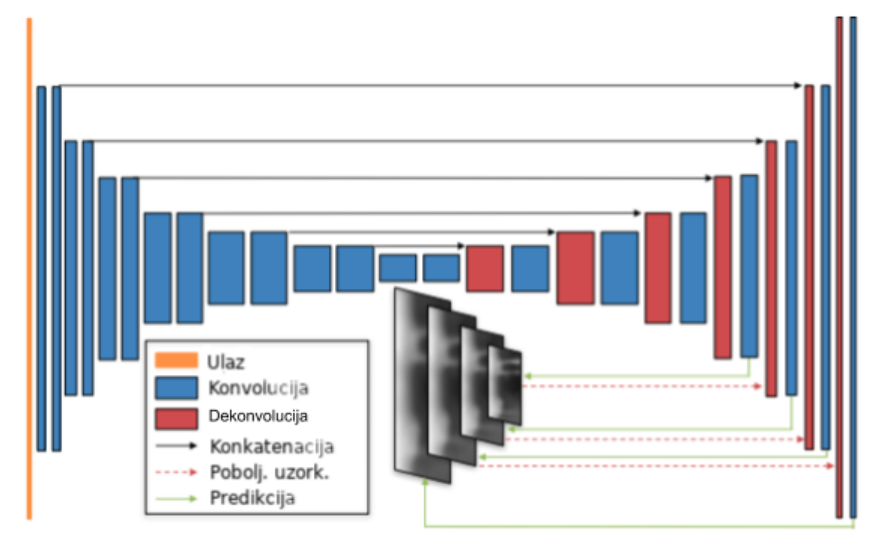
$$Q_{rek} = M_{ij}\left(\frac{\sum_{i,j} |I_{t+1}(i,j) - \hat{I}_{t+1}(i,j)|}{w * h}\right) \tag{4.7}$$

### 4.1.4. Građevne jedinice modela

Građevna jedinica za procjenu dubine ima enkoder-dekoder strukturu (slika 4.5). Kako bi se pripomoglo dekonvolucijama u dekoderskom dijelu modela procjene dubine, dekonvolucijske izlazne mape konkateniraju se s korespodentnim mapama enkodera. Jednostavno se u skup izlaznih mapa dodaju enkoderske mape, odnosno, konkateniraju se po trećoj dimenziji. Na ulazu u mrežu dana je jedna RGB slika (središnja iz niza slika), a na izlazu su četiri dubinske mape različitih skala[3].

[2] U članku [10] veličina ulaznog niza je 3.

[3] Pokazuje se korištenje generiranje četiri dubinske mape različitih skala pomaže pri učenju [10].




**Slika 4.4:** Građevna jedinica za procjenu dubinskih mapa. Izvor: [10].

Neovisno o procjeni dubine, građevna jedinica za procjenu vlastitog gibanja na ulaz dobiva cijeli niz slika. Slike se spajaju u jednu konkatenacijom. Npr., ako je na ulazu 5 slika dimenzija $W$x$H$, konkatenirana ulazna slika bit će dimenzija $5W$x$H$. Takva slika provodi se kroz model prikazan na slici 4.5. Na izlazu se dobiva niz matrica vlastitog gibanja za svaki odnos između središnje i jedne od ostalih slika ($T_{t-2}$, $T_{t-1}$, $T_{t+1}$ i $T_{t+2}$).

Izlazi dvaju građevnih jedinica koriste se za dobivanje prethodno opisanih reprojekcijskih slika. U nastavku opisane su i druge komponente funkcije gubitka koje su se pokazale da pridonose točnosti na testnom skupu (poglavlje 7).

### 4.1.5. Funkcija gubitka

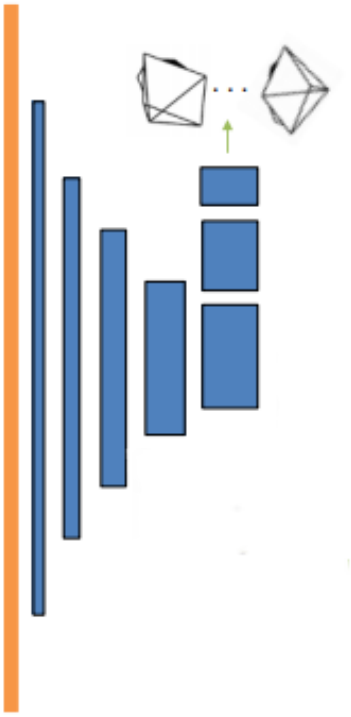
Funkcija gubitka $L$ zbroj je četiriju komponenata:

- pogreške rekonstrukcije,
- pogreške glatkosti dubinske mape,
- pogreške strukturne sličnosti između originalne i rekonstruirane slike (SSIM) te
- pogreške iterativne metode najbližih točaka (ICP) (4.8)[4].

$$L = \sum_{l \in 1...4} \alpha L^l_{rek} + \beta L^l_{glatk} + \gamma L^l_{SSIM} + \delta L^l_{ICP} \tag{4.8}$$

Vrijednosti komponenata funkcije gubitka nisu istog reda veličine pa svaka kompo-

[4]Pogreška iterativne metodu najbližih točaka netrivijalna je komponenta funkcije gubitka pa je zasebno opisana u odsječku 4.2.



**Slika 4.5:** Građevna jedinica procjene vlastitog gibanja. Izvor: [10].

nenta ima svoj pripadni hiperparametar kako bi konačne vrijednosti bile ravnopravne i kako bi se ostvarila pravilna dinamika među komponentama. Indeks $l$ označava skalu na kojoj se pojedina komponenta funkcije računa.

Pogreška glatkosti dubinske mape definirana je funkcijom 4.9. Ona ima ulogu izglađivanja mape tijekom učenja. Parcijalne derivacije mape $\partial_x D^{ij}$ i $\partial_y D^{ij}$ zapravo su razlike susjednih vrijednosti mape po širini, odnosno, dužini. Pripadne eksponencijale ublažavaju učinak izglađivanja na dijelovima na kojima i originalna slika ima grube prijelaze, npr. na rubovima objekata.

$$L_{glatk} = \sum_{i,j} ||\partial_x D^{ij}||e^{-||\partial_x X^{ij}||} + ||\partial_y D^{ij}||e^{-||\partial_y X^{ij}||} \tag{4.9}$$

Neka matrice $X$ i $Y$ (4.13) predstavljaju ulazne slike. Strukturna sličnost između dvije slike $X$ i $Y$ definirana je izrazom 4.11. Elementi $\mu_x$ i $\mu_y$ su srednje vrijednosti ulaznih slika $X$ i $Y$, $\sigma_x$ i $\sigma_y$ su varijance, a $c_1 = 0.01^2$ i $c_2 = 0.03^2$ su konstante. Element $\sigma_{xy}$ zajednička je varijanca slika $X$ i $Y$. Računa se prema izrazu 4.10. U brojniku izraza 4.10 zbrajaju se umnošci vrijednosti piksela $(i, j)$, u nazivniku je ukupan broj piksela i od tog se razlomka oduzima umnožak srednjih vrijednosti ulaznih slika $X$ i $Y$.

U slučaju potpune strukturne sličnosti, vrijednost 4.11 bit će 1. S obzirom da je izraz potrebno minimizirati, pogreška strukturne sličnosti definirana je kao 4.12. Matrica $M_t^{ij}$ je maska preklapanja (4.6).

$$\sigma_{xy} = \frac{\sum_{ij} x_{ij} y_{ij}}{w * h} - \mu_x \mu_y \tag{4.10}$$

$$SSIM(X, Y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x + \sigma_y + c_2)} \tag{4.11}$$

$$L_{SSIM} = \sum_{ij} [1 - SSIM(\widetilde{X}_t^{ij}, X_t^{ij})] M_t^{ij} \tag{4.12}$$

Izrazi (4.14a, 4.15a i 4.16a) u nastavku računaju svaku od dosad definiranih komponenata funkcije gubitka, kao i ukupni gubitak (4.17a). Odabrane su vrijednosti hiperparametara $\alpha = 0.85$, $\beta = 0.1$, $\gamma = 0.05$, a $\delta$ nije postavljena. Pogreška glatkosti matrice $A$ izračunata je, zbog jednostavnosti, bez pripadnih eksponencijala. Za pogrešku strukturne sličnosti unaprijed su izračunate vrijednosti $\mu_x = 0.3944$, $\mu_y = 0.4444$, $\sigma_x = 0.061393$, $\sigma_y = 0.079175$ i $\sigma_{xy} = 0.066951$.

$$X = \begin{bmatrix} 0.1 & 0.4 & 0.15 \\ 0.8 & 0.25 & 0.65 \\ 0.15 & 0.35 & 0.7 \end{bmatrix}, Y = \begin{bmatrix} 0.15 & 0.5 & 0.25 \\ 0.8 & 0.15 & 0.8 \\ 0.2 & 0.3 & 0.85 \end{bmatrix} \tag{4.13}$$

$$L_{rek} = \frac{|X - Y|}{9} \tag{4.14a}$$

$$= \frac{\left| \begin{bmatrix} 0.1 & 0.4 & 0.15 \\ 0.8 & 0.25 & 0.65 \\ 0.15 & 0.35 & 0.7 \end{bmatrix} - \begin{bmatrix} 0.15 & 0.5 & 0.25 \\ 0.8 & 0.15 & 0.8 \\ 0.2 & 0.3 & 0.85 \end{bmatrix} \right|}{9} \tag{4.14b}$$

$$= 0.0833 \tag{4.14c}$$

$$L_{glatk}(X) = \frac{||\partial_x X|| + ||\partial_y X||}{9} \tag{4.15a}$$

$$= \frac{(|0.4 - 0.1| + ... + |0.7 - 0.35|) + (|0.5 - 0.15| + ... + |0.85 - 0.3|)}{9} \tag{4.15b}$$

$$= 0.5 \tag{4.15c}$$

$$L_{SSIM} = 1 - \frac{(2 * 0.3944 * 0.4444 + 0.01^2)(2 * 0.066951 + 0.03^2)}{(0.3944^2 + 0.4444^2 + 0.01^2)(0.061393 + 0.079175 + 0.03^2)} \quad (4.16a)$$

$$= 1 - 0.9461 \quad (4.16b)$$

$$= 0.0539 \quad (4.16c)$$

Konačno,

$$L = 0.85 * 0.0833 + 0.1 * 0.5 + 0.05 * 0.0539 \quad (4.17a)$$

$$= 0.1235. \quad (4.17b)$$

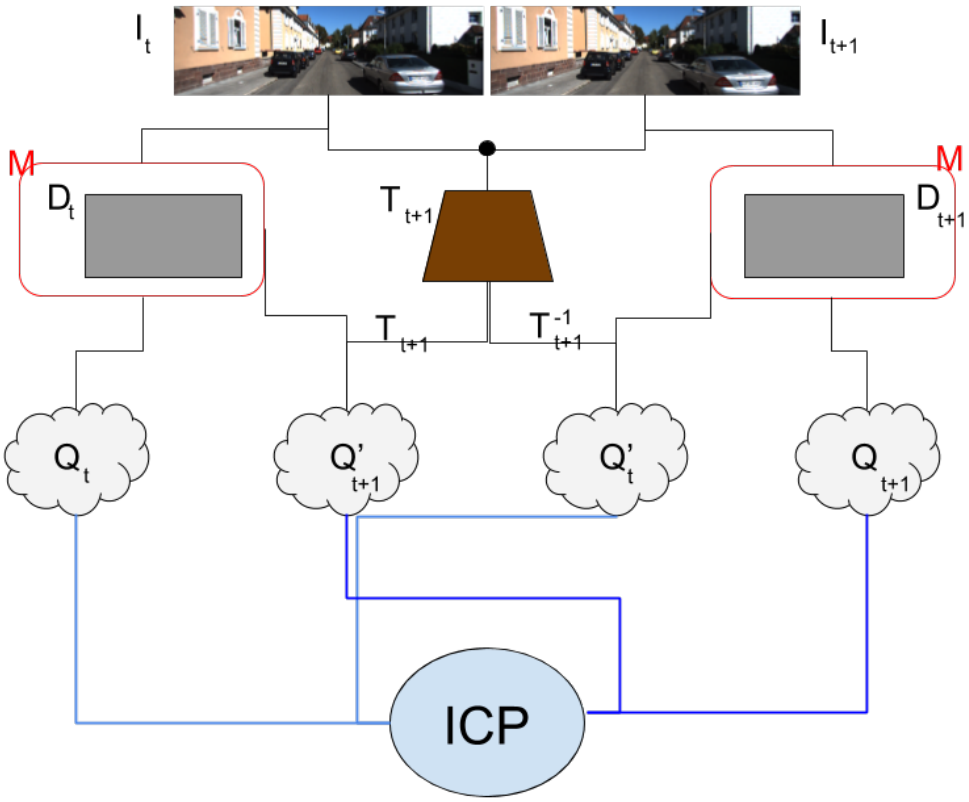
## 4.2. Iterativna metoda najbližih točaka u funkciji gubitka

U nastavku opisani su detalji članka [3] nastavnog na [10]. Ideja članka je otići korak dalje od procjene dubinske mape. Iz poglavlja 2 jasno je da se iz dubinske mape $D_t$, množenjem s inverzom kalibracijske matrice $K^{-1}$, dobiva oblak točaka $Q_t$ koji predstavlja dubinsku mapu u 3D prostoru. Množeći oblak točaka slijeva transformacijskom matricom $T_{t+1}$, dobiva se oblaka točaka u budućem trenutku $Q'_{t+1}$. S druge strane, za dubinsku mapu sljedeće slike $D_{t+1}$, dobiva se oblak točaka $Q_{t+1}$. Množeći inverzom transformacijske matrice $T_{t+1}^{-1}$, dobiva se oblak točaka za sadašnji trenutak $Q'_t$. Oblaci $Q'_t$ i $Q_t$, odnosno, $Q'_{t+1}$ i $Q_{t+1}$ preklapaju se[5] (slika 4.6). Razlike između korespodentnih točaka preklopljenih oblaka su pogreške i koriste se u funkciji pogreške. Ideja je preklopiti oblake točaka $\widetilde{Q}_t$ i $Q_t$ te $\widetilde{Q}_{t+1}$ i $Q_{t+1}$ i njihove razlike iskoristiti za funkciju gubitka.

### 4.2.1. Optimalno preklapanje skupa točaka u zatvorenoj formi

Zadana su dva skupa točaka $X = \{x_1, ..., x_n\}, x_i \in \mathbb{R}^3$ i $P = \{p_1, ..., p_n\}, p_i \in \mathbb{R}^3$. Traži se optimalna translacija $t$ i rotacija $R$ koja minimizira 4.18. Znajući korespodentne točke skupova $X$ i $P$, transformacija se može izračunati u zatvorenoj formi [11].

[5]Nad jednim oblakom obavlja se takva transformacija koja maksimalno dobro preklapa oblake točaka. To radi algoritam ICP.




**Slika 4.6:** Model s naglaskom na ICP komponentu.

$$E(R,t) = \frac{1}{N_p}\sum_{i=1}^{N_p} ||x_i - Rp_i - t||^2 \tag{4.18}$$

Prvi korak je svakako pronaći centar mase skupa. Centar mase računa se prema izrazu 4.19. Taj iznos oduzima se od svake točka skupa (4.20). Identičan postupak provodi se i nad skupom $P$.

$$\mu_x = \frac{1}{N_x}\sum_{i=1}^{N_x} x_i \tag{4.19}$$

$$X' = \{x_i - \mu_x\} = \{x_i'\} \tag{4.20}$$

Neka je matrica $W = \sum_{i=1}^{N_p} x_i' p_i'^{\top}, W \in \mathbb{R}^{3x3}$. Singularna dekompozicija matrice (SVD) definirana je jednakošću 4.21, gdje su $U, V \in \mathbb{R}^{3x3}$ unitarne matrice i $\sigma_1 > \sigma_2 > \sigma_3$ vlastite vrijednosti matrice $W$. Unitarna matrica je ona matrica čija je konjugirana transpozicija ujedno i njen inverz [12], odnosno, $U^*U = UU^* = I$.

$$W = U \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix} V^{\top} \tag{4.21}$$

Minimum funkcije 4.18 za danu matricu $W$ ranga 3 je jedinstven (4.22c). Optimalna matrica rotacija dobiva iz 4.22a, a translacijski vektor iz 4.22b.

$$R = UV^{\top} \tag{4.22a}$$

$$t = \mu_x - R\mu_p \tag{4.22b}$$

$$E(R,t) = \sum_i (||x_i'||^2 + ||p_i'||^2) - 2(\sigma_1 + \sigma_2 + \sigma_3) \tag{4.22c}$$

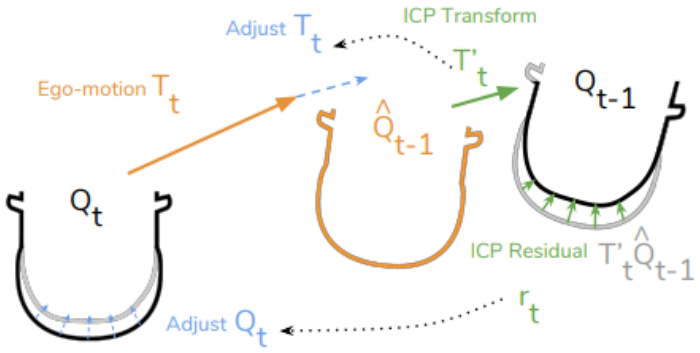
### 4.2.2. Iterativna metoda najbližih točaka (ICP)

Ne znajući korespodentne točke između skupova, rješenje ne postoji u zatvorenoj formi. Koristi se iterativni postupak koji u prvom koraku procjenjuje korespodentne točke, a u drugom nalazi transformaciju koja preklapa točke s obzirom na procijenjene korespodencije. Ovaj se postupak naziva iterativna metoda najbližih susjeda (ICP, eng. *iterative closest point*). Za traženje korespodencija može se koristiti algoritam najbližih susjeda. Pseudokod ICP algoritma dan je u nastavku.

**Algoritam 2:** ICP algoritam

```
fun nadi_najbolju_transformaciju(X, P):
    μ_x = μ(X, os=0), μ_p = μ(P, os=0)
    X' = X - μ_x, P' = P - μ_p
    H = (X').T * P
    U, S, V_t = SVD(H)
    ako det(R) < 0 onda
        V_t[1] *= -1
    kraj
    R = U * V_t.T
    t = (P').T - R * (X').T
    T = [R|t]
    vrati (T)
fun icp(X, P, max_iter, tol):
    preth_pogreska = 0.0
    T = I
    dok broj_iteracija < max_iter čini
        udaljenosti, permutacija = najblizi_susjedi(X, P)
        T' = nadi_najbolju_transformaciju(X, P[permutacija])
        X = T' * X, T = T' * T
        ako |μ(udaljenosti) - preth_pogreska| < tol onda
            prekini petlju
        kraj
        preth_pogreska = μ(udaljenosti)
    kraj
    vrati (T, udaljenosti)
```

Argumenti funkcije ICP su dva skupa točaka $X$ i $P$, maksimalni broj iteracija i minimalna tolerancija na promjenu pogreške između iteracija. Algoritmom najbližih susjeda procjenjuje se korespodencija između skupova točaka, vraćaju se udaljenosti korespodentnih točaka i permutacijski skup koji određuje korespodencije. Permutacijski skup argument je unutarnje funkcije za pronalazak najbolje transformacije - matrice $T$. Unutarnja funkcija određuje centroide i oduzima ih od skupova točaka. Provodi SVD i dobiva transformacijsku matricu. Usput se provjerava determinanta matrice R u slučaju da je optimalna transformacija za $180°$ zakrenuta od one izravno dobivene SVD-om. Matrica $T$ vraća se iz unutarnje funkcije i množi skupom točaka $X$. Ukoliko je prosječna udaljenost različita od prethodne prosječne udaljenosti u prethodnom koraku, postupak se zaustavlja. U suprotnom, prosječna udaljenost "prethodnog"




**Slika 4.7:** Reziduali i transformacije ICP-a. Izvor: [3].

koraka postaje trenutna udaljenost i postupak se nastavlja dok se ne dostigne uvjet zaustavljanja. Na kraju, vraća se procijenjena matrica transformacije $T$ i udaljenosti korespodentnih točaka za tu transformaciju.

### 4.2.3. Aproksimacija derivacije kombinatornog algoritma

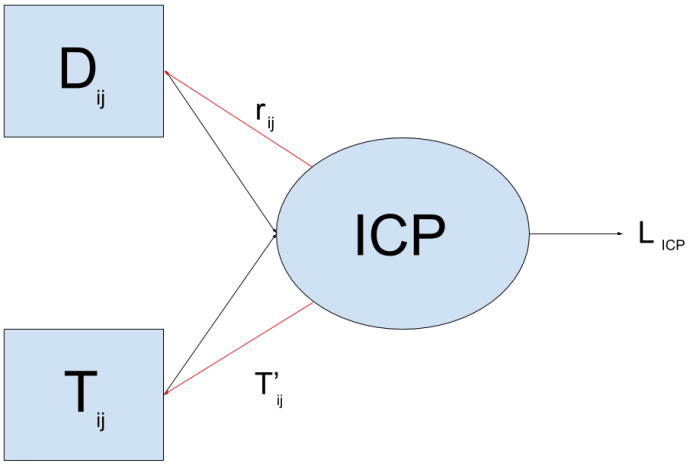
ICP je kombinatorni algoritam [13] i nije derivabilan [3]. Gradijenti koji se propagiraju unatrag se stoga aproksimiraju. Ideja je da, nakon preklapanja, razlika korespodentnih točaka bude minimalna. Te se razlike nazivaju rezidualima i definirane su izrazom 4.23, gdje $c(\cdot)$ označava poredak točaka u skladu s korespodentnim poretkom. Reziduali su posljedica pogrešne procjene dubinske mape pa se propagiraju unatrag samo tom granom dubokog modela (slika 5.1).

$$r^{ij} = \widetilde{T}^{-1} Q_t^{ij} - Q_{t+1}^{c(ij)} \tag{4.23}$$

Osim reziduala, optimalna transformacijska matrica $T$ dobivena ICP-om služi za korekciju procjene vlastitog gibanja $T_{t\to t+1}$, tj. $T_{t+1\to t}$. Odstupanje od idealne transformacije može se izraziti kao 4.24. Pogreška iterativne metode najbližih točaka je konačno dana s 4.25. Propagacija unaprijed i unatrag kroz ICP čvor prikazana je na slici 4.8.

$$\Delta T = ||\widetilde{T}_{t+1} - I|| \tag{4.24}$$

$$L_{ICP} = ||\underbrace{\widetilde{T}^{-1} Q_t^{ij}}_{\hat{Q}_{t+1}^{ij}} - Q_{t+1}^{c(ij)}|| + ||\widetilde{T}_{t+1} - I|| \tag{4.25}$$




**Slika 4.8:** Propagacija unaprijed i unatrag kroz ICP čvor.

# 5. Skupovi podataka za učenje

Duboko učenje snažno se oslanja na podatke. Često su potrebne tisuće, pa i milijuni slika za postizanje prihvatljive točnosti i preciznosti na testnom skupu. Slike su nestrukturirani podaci i modelu su jedina sugestija oznake koje se nalaze na izlazu u slučaju nadziranog učenja. Štoviše, model opisan u ovom radu primjer je nenadziranog učenja i izvorni ulazni podaci su jedina informacija na koju se može osloniti tijekom učenja.

## 5.1. Poznati skupovi slika

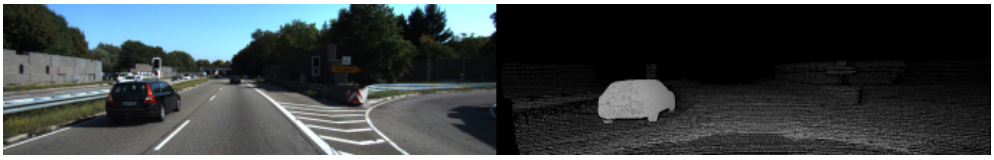
Modeli [10] i [3] učeni su nad poznatim skupovima podataka KITTI i Cityscapes ([14], [16]). To su skupovi podataka urbanih naselja uslikani vožnjom automobilom i danas su jedni od najpopularnijih skupova za učenje i validaciju algoritama autonomne vožnje.

### 5.1.1. Skup podataka KITTI

Podatkovni skup KITTI nastao je 2015. Sadrži više od 45000 slika za treniranje. KITTI sadrži stereo parove slika - za svaki vremenski trenutak po dvije slike, iz lijeve i desne kamere rektificiranog para. Za oko 30% parova slika iz skupa postoje informacije o disparitetima dobivene laserskim uređajem. Na slici 5.1 primjer je slike iz skupa podataka i pripadne mape dispariteta. Za skup slika za koje postoje laserski dobiveni dispariteti, mape dispariteta su rijetke (poglavlje 2). Naknadnom obradom dispariteti su proguščeni ubacivanjem 3D modela automobila u odgovarajući oblak točaka[1]. Mape dispariteta imaju vrijednost nula na mjestima gdje nisu poznati dispariteti.

Stereo par kamera nalazi se na krovu automobila te slike prikazuju scene iz vožnje u gradskim sredinama. Skup podataka sniman je kroz nekoliko dana. S obzirom da

[1]Ovo implicira kako je ključno da stereo postupak bude precizan na automobilima kako bi ostvario visoku točnost [15].



**Slika 5.1:** Primjer slike i pripadne disparitetne mape podatkovnog skupa KITTI.

KITTI sadrži neke stvarne disparitete slika, uz skup podataka dane su i kalibracijske matrice tih slika. Točnije, postoji nekoliko kalibracijskih matrica ovisno o dobu dana u koje su slike prikupljane.

### 5.1.2. Skup podataka Cityscapes

Cityscapes skup snimljen je 2016. u 50 njemačkih gradova i sadrži oznake piksela na razini instance[2] za semantičku segmentaciju (primjer članka semantičke segmentacije na razini instance [26]). U skup podataka je 5000 fino označenih slika i još 20000 grublje označenih. Cityscapes nema javno označene disparitete mape pa nije pogodan za validaciju tijekom učenja.

Kao i skup KITTI, Cityscapes također prikazuje urbane scene. Zbog toga se Cityscapes često koristi za fino podešavanje modela (eng. *fine-tuning*) nakon baznog učenja nad KITTI skupom. Bilo bi zanimljivo napraviti obrnuti eksperiment, odnosno, nad Cityscapes-om provesti bazno učenje, a nad KITTI skupom fino podešavanje modela. Prepreka u tome je što Cityscapes nema stvarne oznake dispariteta. Tome bi se, ipak, moglo doskočiti tako da se tijekom učenja na Cityscapesu kvaliteta provjerava testnim skupom KITTI.

## 5.2. Priprema podataka

Slike na ulazu u model posebno su pripremljene i uređene. Ključno je za model da su slike poredane kronološki i da postoji područje preklapanja slika kako bi se na što većem broju piksela ocijenila funkcija gubitka i time maksimalno iskoristila informacija dostupna iz niza slika. Ulazne slike poduzorkuju se na veličinu 128x416.

### 5.2.1. Grupiranje slika u kronološke nizove

Neka je ulazni niz duljine 3. Na slici 5.2 s lijeve strane prikazano je prvih pet slika skupa za učenje. Prve tri slike spajaju se u prvi ulazni niz. Druga, treća i četvrta slika

[2]Svaki objekt na slici istaknut je zasebno, npr. svaki automobil ili pješak.

**Slika 5.2:** Priprema ulaznih nizova slika.

čine drugi niz, itd... Za razliku od prikazane slike 5.2, nizovi se spajaju u slike po širini, ne visini, tako da se dobivaju ulazne slike dimenzija $H * (3 * W)$. Ulaznih slika je $(N - 1)$, gdje je $N$ broj slika skupa za učenje. U svakoj epohi nizovi se slučajno miješaju. To se pokazuje kao dobra regularizacijska tehnika jer se model ne veže uz poredak slika i time se povećava generalizacijska moć.

### 5.2.2. Proširivanje skupa podataka i predobrada slika

Proširivanje skupa podataka provodi se zanimljivom transformacijom ulaznog niza slika. Transformacija se odjednom provodi nad cijelom konkateniranom slikom, ali zbog jednostavnosti u nastavku je opisana transformacija nad jednom, odvojenom slikom $I_i$ dimenzija $W \cdot H$. Odabire se broj iz slučajne razdiobe između $1$ i $1.15$ po obje osi slike $s'_x$ i $s'_y$. Visina i širina skaliraju se odabranim brojem i dobiva se slika dimenzija $(s'_x \cdot W, s'_y \cdot H)$. Slika je sad uvećana. Kako bi se zadržale originalne dimenzije, slučajno se odabire okvir veličine $W \cdot H$ nad uvećanom slikom. Okvir je definiran odmakom od gornjeg lijevog ruba $(c'_x, c'_y)$, a donji desni rub je onda $(c'_x + W, c'_y + H)$.

Slika $I_i$ ima pripadnu kalibracijsku matricu $K$ (koja je zajednička većem broju

slika, npr. u 5.1.1). Transformacije nad slikom moraju se odraziti i na kalibracijsku matricu. Konkretno, fokalni faktori intrinzične matrice $f_x$ i $f_y$ množe se s $s'_x$ i $s'_y$, a koordinatama središta se dodatno oduzima $c'_x$, odnosno, $c'_y$. Nova intrinzična matrica slike $I_i$ dana je s 5.1.

$$K_i = \begin{bmatrix} f_x \cdot s_x & s & c_x \cdot s_x - c'_x \\ 0 & f_y \cdot s_y & c_y \cdot s_y - c'_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5.1}$$

Iz slike $I_i$ dobivaju se 4 ulazne slike različitih skala. Pokazuje se da provođenje čitavog postupka učenja nad točno 4 različite skale daje najbolje rezultate. Prije ulaska u model, slike se predobrađuju s $I_i^{ij} = I_i^{ij} \cdot 2. - 1.$

# 6. Implementacija modela u programskom okviru TensorFlow

TensorFlow [22] je Google-ov programski okvir s kraja 2015. godine koji je popularizirao duboko učenje svojom tadašnjom jednostavnošću i performansama[1]. Napisan je u C++-u za izvođenje na NVIDIA-nim grafičkim karticama, odnosno, hardveru koji podržava CUDA programiranje. TensorFlow se oslanja na statičke računske grafove i diferencijabilno programiranje. Sav izvorni kod dostupan je na GitHubu pod Apache 2.0 licencom. Iza njega je ogromna skupina ljudi i napisano je nekoliko programskih okvira više razine koji pomažu u radu s TensorFlow-om (npr. *slim*). Izvorna inačica TensorFlowa pruža sučelje za više programskih jezika, npr., Python i C++, a postoje i implementacije u drugim jezicima, npr. TensorFlowJS u Typescriptu.

## 6.1. Učenje nad grupama

Učenje nad grupama tehnika je paralelnog izvođenja većeg broja iteracija odjednom. Neka je veličina grupe jednaka 4. To znači da se na ulaz stavljaju 4 (konkatenirane) slike i paralelno se provode kroz model. Na izlazu iz modela za procjenu dubine su 4 dubinske mape (na četiri različite skale!), a na izlazu iz modela za procjenu vlastitog gibanja su $4 \cdot 2$ matrice vlastitog gibanja (jedna za prošli, a jedna za budući vremenski trenutak).

Učenje nad grupama počelo se koristiti pojavom grafičkih procesora (GPU-ova). Za razliku od običnih procesora koji naredbe izvode proceduralno (slijedno), grafički procesori omogućuju da se (određene) operacije izvode paralelno i time znatno smanji njihova složenost. Npr., zbrajanje vektora na GPU izvodi se u vremenskoj složenosti $O(1)$, a naivno množenje matrica u složenosti $O(n^2)$. Ovu optimizaciju omogućuje posebna arhitektura grafičkih procesora.

[1] S vremenom se pokazalo da je Google odabrao pogrešan put oslanjajući se na Theano umjesto na Torch.


**Slika 6.1:** Arhitektura Nvidia-e 1080.

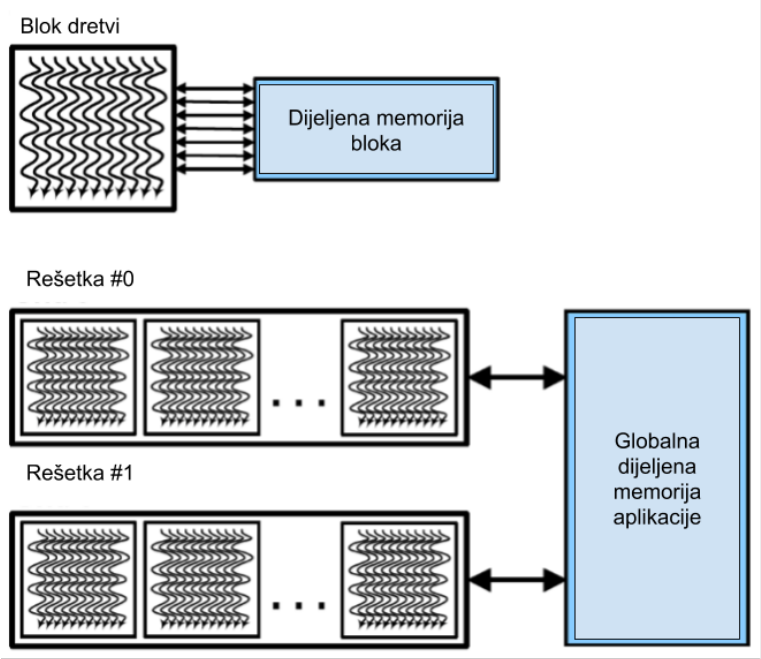
### 6.1.1. Fizička i logička organizacija GPU-a

Na slici 6.1 primjer je arhitekture relativno jake grafičke kartice, NVidia-e GeForce 1080, koja je upotrebljiva za napredne primjene dubokog učenja. Ono na što je važno obratiti pozornost je broj blokova označenih kraticom "SM" - višeprocesorskih blokova (eng. *streaming multiprocessors*). To su blokovi koji sadrže svaki po 2x2 reda zelenih ćelija. Usput, višeprocesorski blokovi organizirani su u grafičke procesorske grozdove (eng. *graphical processing clusters* - GPC), a svaki je GPC upravljan dvama memorijskih upravljačima prikazanima na rubovima slike.

Unutar svakog višeprocesorskog bloka (slika 6.2) nalaze se 4 procesorska bloka sa po 4x8 jezgara. Četiri fizička procesorska bloka omogućuju izvršavanje (doslovno) paralelne operacije na 4 dretve, u svakoj jezgri po jedna, bez posebnog redoslijeda. Npr., istovremeno se izvršava *Operacija 7* na *jezgri 15*, potom *Operacija 2* na *jezgri 5*, itd... U slučaju GeForce 1080, ta sekvenca traje 32 jedinice vremena dok se ne izvrše sve predviđene operacije tog logičkog bloka. Na razini čitavog grafičkog procesora, izvršeno je $32 * 80 = 2560$ operacija. One se smatraju paralelnim operacijama u fizičkom smislu jer je broj 32 konstanta na razini fizičke arhitekture.

U kontekstu logičke organizacije, 2560 fizičkih jezgara čini logički blok, a blokovi


**Slika 6.2:** Višeprocesorski blok s 4 procesorska bloka po 32 jezgre.



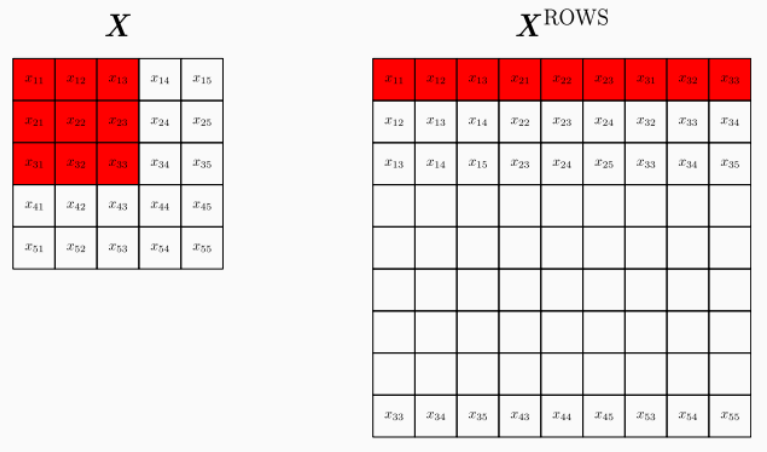

**Slika 6.3:** Logička razina GPU-a. Svaki blok dretvi ima svoju dijeljenu memoriju. Blokovi se raspoređuju u rešetke (engl. *grid*), gdje svaka rešetka komunicira s globalnom dijeljenom memorijom aplikacije.

su organizirani u logičke rešetke (slika 6.3). Veličina logičke rešetke ograničena je fizičkom veličinom grafičkog procesora. Neka se, npr., zbrajaju dva vektora veličine 26000 *float*-ova. Složenost te operacije na GPU-u jest $\lceil 26000/2560 \rceil = 11 \sim O(1)$. Uz pravilno korištenje logičke rešetke i blokova prilikom pisanja koda za GPU (u CUDA-i), moguće je i praktično ostvariti ovu teoretsku složenost. Srećom, TensorFlow ove detalje sakriva od programera koji razvija model dubokog učenja i većinom ne stvara poteškoće.

Slanje podataka između računalnih komponenata (SSD, RAM, VRAM) obično traje najduže u čitavom procesu pa je najpogodnije poslati što više podataka odjednom u radnu memoriju grafičke kartice. Upravo je to motivacija za korištenje učenja nad grupama. U praksi se isproba nekoliko većih vrijednosti veličine grupe dok se od TensorFlowa ne dobije poruka o pogrešci zbog prekoračenja veličine memorije grafičke kartice.

### 6.1.2. Paralelizacija konvolucije na GPU

Neka je ulazna slika veličine 5x5 piksela i jezgra veličine 3x3 i neka su ostale dimenzije zbog jednostavnosti zanemarene. Konvolucija se na GPU optimalno može izvesti




**Slika 6.4:** Optimalna izvedba dijela konvolucije na GPU.

prema prikazu 6.4. Crvenom bojom označeni su elementi koje množi jezgra konvolucije. Množenje retka jezgrom svakako se može izvesti u složenosti $O(1)$. Štoviše, elementi se na GPU mogu posložiti na takav način da se čitavo matrično množenje odvije u jednom fizičkom trenutku. Množenje prvog retka zauzima 9 operacija, drugog još 9, a ukupno je 9 redaka, pa će se ukupno izvršiti $9 * 9$ operacija, što svakako stane u jezgre GeForce 1080-ice.

Valja ipak primijetiti redundantnost u prostornoj složenosti. Umjesto $5 * 5$ elemenata, na GPU dolazi $9 * 9$ elemenata. Kolika će redundantnost biti ovisi o odabranim parametrima konvolucije. Npr. da je korak konvolucije jednak 2, prozor bi se svaki put pomaknuo za dva elementa, pa bi matrica slike na GPU bila dimenzija $4 * 9$. Sve je to u praksi zanemarivo s obzirom na poboljšanje vremenske složenosti, ali objašnjava zašto duboki modeli zahtijevaju veliku količinu radne memorije.

## 6.2. Diferencijabilno programiranje

Diferencijabilno programiranje je programska paradigma koja, između ostalog, generalizira unazadnu propagaciju gradijenata dubokog učenja [17]. Duboki model aciklički je graf čije su jedinice funkcije. Funkcije uzimaju ulazne argumente, obrađuju ih te prosljeđuju rezultat unaprijed kroz model. Prilikom ažuriranja parametara unazadnom propagacijom, računa se njena derivacija. U TensorFlowu, jedinice modela su čvorovi računskog grafa kojeg program izvodi. Svaka jedinica mora imati definirano ponašanje u oba smjera. Tek tada je moguće za dani računski graf provesti, tzv. automatsku diferencijaciju (AD). No, čvorovi računskog grafa nisu samo jedinice dubokog modela. Čvorovi su sve operacije potrebne za unaprijednu i unazadnu propagaciju

kroz model, učitavanje modela te računanje funkcije gubitka. U diferencijalnom programiranju, računski graf općenito je računalni program parametriziranih funkcijskih blokova [18].

## 6.3. Nadjačavanje gradijenata u TensorFlowu

TensorFlow pruža svoje implementacije slojeva konvolucije, sažimanja, potpuno povezanog sloja, ReLU funkcije, *softmax*-a i mnoge druge praktično korisne čvorove. Implementacija ICP-a ne postoji u TensorFlowu pa je nekako potrebno ugraditi njegovu funkcionalnost. Jedna je mogućnost proširiti jezgru TensorFlow-a implementacijom algoritma te kompajlirati proširenu verziju. TensorFlow se kompajlira alatom *bazel* i pokazuje se kao vrlo zahtjevan zadatak. Druga je mogućnost u grafu stvoriti čvor koji propagacijom unaprijed preslikava ulaze na izlaz iz čvora (funkcija identiteta), a propagacijom unatrag ulaznim granama čvora pridijeljuje gradijente opisane u 4.2.3. Druga se mogućnost naziva nadjačavanje gradijenata i to je odabrana metoda implementacije ICP-a. U nastavku je pokazano nadjačavanje gradijenta na jednostavnom primjeru kako bi se uočili svi bitni detalji implementacije. Potom je opisana stvarna implementacija koja "zamata" Python implementaciju ICP-a u računski graf i pridijeljuje aproksimacije gradijenata.

### 6.3.1. Izvedba na jednostavnom primjeru

Neka je dana varijabla 'x' početne vrijednosti 3.0. Ideja je provesti varijablu kroz funkciju identiteta u prolasku unaprijed i ručno pridijeliti pet puta veći gradijent pri prolasku unatrag. Za usporedbu, stvorena je i druga grana grafa koja također provodi funkciju identiteta unaprijed, a unatrag ugrađeni gradijent iz TensorFlow-a (također identitet). Očekuje se da je će ugrađeni gradijent i uvećani gradijent biti u omjeru 1:5. Izvorni kod koji se može pokrenuti s TensorFlow-om 1.7 dan je u nastavku.

```
import tensorflow as tf

ulaz = tf.get_variable('x', dtype=tf.float32, initializer=[3.0])

@tf.custom_gradient
def cvor_nadjacanog_gradijenta(x):
  def grad(dy):
    return 5.0 * dy
  return tf.identity(x), grad
```

```
izlaz_test = cvor_nadjacanog_gradijenta(ulaz)
gradijent_test = tf.gradients(izlaz_test, ulaz)

izlaz = tf.identity(ulaz)
gradijent = tf.gradients(izlaz, ulaz)

with tf.Session() as sess:
  sess.run(tf.global_variables_initializer())
  print("Uvecani gradijent: ", sess.run(gradijent_test)[0])
  print("Ugradeni gradijent: ", sess.run(gradijent)[0])
```

Ključni dio koda je dekorator *tf.custom_gradient* koji funkciji *clip_grad_layer* dodaje određenu implicitnu funkcionalnost. Od funkcije se očekuje da je njena druga povratna vrijednost funkcija koja definira gradijent. TensorFlow prilikom automatske diferencijacije zove tu funkciju (*grad*) s argumentom $dy$ koji, u ovom slučaju, ima istu vrijednost kao i ulaz u čvor.

Blok ključne riječi *with* je upravitelj kontekstom (eng. *context manager*) varijable *sess*. Mehanizam upravitelja kontekstom vrlo je sličan i može se svesti na mehanizam dekoratora u Pythonu [19]. Upravitelj stvara varijablu *sess=tf.Session()* i brine se o tome da se na kraju bloka izvedu sve potrebne naredbe, npr. zatvaranje opisnika datoteka i sl. Funkcija *sess.run* evaluira dane čvorove i vraća potrebnu *numpy* strukturu. U ovom slučaju, to je broj tipa *float32*.

### 6.3.2. Nadjačavanje izlazom ICP algoritma

Neka su procijenjene dubinske mape $D_t$ i $D_{t+1}$ te jedna od matrica vlastitih gibanja, npr. $T_{t\to t+1}$. Iz tih procjena stvoreni su oblaci točaka $Q_t$ i $Q_{t+1}$ koji su proslijeđeni kao argumenti skupa s transformacijom $T_{t\to t+1}$. Točnije, proslijeđena je samo $x$ komponenta translacije[2]. Iz trenutnog oblaka točaka $Q_t$ i danom procjenom transformacije $t_x$ dobiva se procjena oblaka točaka u budućem trenutku $Q'_{t+1}$ koji se preklapa s $Q_{t+1}$. Za implementaciju samog ICP-a, koristi se *Open3D* biblioteka u Pythonu [21] koja, između ostalog, implementira efikasan paralelni ICP algoritam.

```
from open3d import *
```

[2]Pri unazadnom prolasku dobivaju se razlike u komponentama $r_x$, $r_y$ i $r_z$. Iz poglavlja 2 poznato je da se rotacija primijenjuje množenjem rotacijskom matricom. Gradijenti rotacijskog dijela matrice, koji bi se primijenili operacijom oduzimanja, stoga nebi bili ispravni. Dodatno se pokazuje da su, za ovaj zadatak i skupove podataka nad kojima se uči, komponente $t_y$ i $t_z$ translacije zanemarive i, štoviše, pridonose lošijem rezultatu učenja.

```
def _registration_icp(tren_oblak, sljedeci_oblak, tx):
    reg = registration_icp(tren_oblak,
                           sljedeci_oblak,
                           tx,
                           TransformationEstimationPointToPlane())
    return reg.residuals, reg.transformation[0][0]

@tf.custom_gradient
def icp_op(tren_oblak, sljedeci_oblak, tx):
    def grad(reziduali, tx_):
        return reziduali, tx_
    reziduali, tx_ = tf.py_func(_registration_icp,
                                [tren_oblak, sljedeci_oblak, tx],
                                [tf.float32, tf.float32])
    return reziduali, tx_, grad

def izracunaj_icp_gubitak(tren_oblak, sljedeci_oblak, tx):
    reziduali, tx_ = icp_op(tren_oblak, tren_oblak, tx)
    return tf.abs(reziduali) + tf.abs(tx_)
```

Funkcija *izracunaj_icp_gubitak* implementira čitavu funkcionalnost ICP dijela zadatka. Zove se funkcija *icp_op* koja je dekorirana, odnosno, označena kao funkcija od koje se očekuje podrška za nadjačavanje gradijenta. Funkcija *tf.py_func* zamata funkciju *_registration_op*, koja nije pisana za TensorFlow, u TensorFlow čvor. Drugi je argument funkcije *tf.py_func* lista argumenata funkcije *_registration_op*. Treći je argument lista tipova povratnih vrijednosti funkcije *tf.py_func*. Njih je potrebno eksplicitno zadati jer se zamata proizvoljna funkcija i nije automatski moguće odrediti koji će biti povratni tipovi u kontekstu TensorFlow-a. Konačno, poziva se ICP algoritam nad *tren_oblak* i *sljed_oblak*. Transformacija $tx$ inicijalna je transformacija koja se primijenjuje na *tren_oblak*. Koristi se specifična implementacija ICP-a koja projicira točke na površinu umjesto da traži korespodenciju među točkama [20]. Pokazuje se da ta inačica algoritma daje bolje rezultate [3]. Prilikom propagacije unatrag, jednostavno se vraćaju iste vrijednosti koje su dane na izlazu pri propagaciji unaprijed. Vrijednost funkcije gubitka modificirana je u odnosu na 4.25. Funkcija *izracunaj_icp_gubitak* vraća vrijednost funkcije gubitka (skalar). Valja istaknuti da funkcija *_registration_icp* nije redundantna. Ona je potrebna kako bi se iz funkcije *tf.py_func* vratila dva argumenta i njima pridijelili tipovi *tf.float32*.

# 7. Rezultati eksperimenata

Provedena je kvalitativna i kvantitativna usporedba dvaju modela. Kvalitativna usporedba komentira dubinske mape triju odabranih slika iz skupa KITTI. U kvalitativnoj usporedbi dani su konkretni brojevi dobiveni metrikama koje se standardno koriste za evaluaciju dubinskih mapa. Opisane su metrike i način na koji se evaluiraju mape.

Iterativna metoda najbližih susjeda nije implementirana u postupku učenja modela iz poglavlja 4 pa su uspoređeni modeli prema korištenoj funkciji pogreške glatkosti dubinske mape (pododsječak 4.1.5). Model koji koristi eksponencijale (vezan uz članak [3]) daje superiornije rezultate od modela koji ih ne koristi (model iz članka [10]).

## 7.1. Kvalitativna usporedba dubinskih mapa

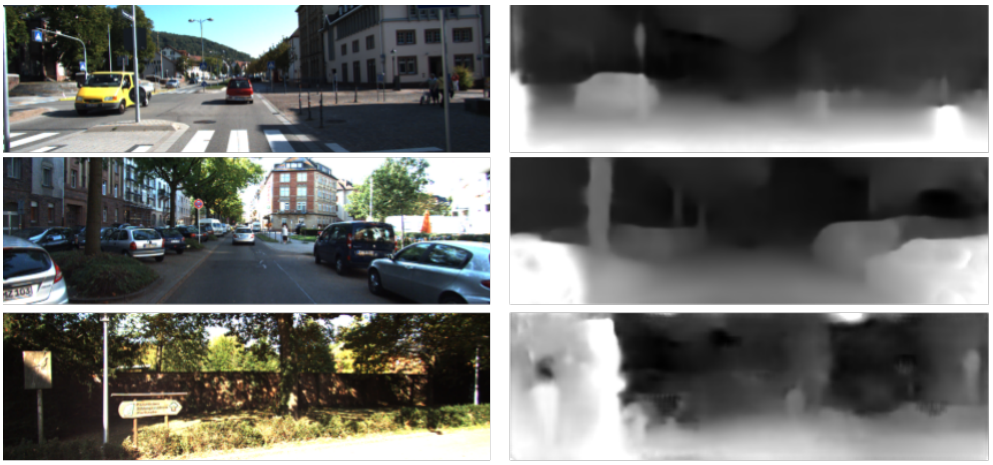
Na slici 7.1 dani su primjeri slika i pripadne procjene dubinskih mapa modela učenog nad KITTI skupom bez korištenja eksponencijala. Na prvoj slici, model relativno dobro prepoznaje bliske parkirane automobile s obje strane ceste. S druge strane, automobil koji se nalazi na otprilike 25 metara udaljenosti od promatrača uopće ne prepoznaje kao relativno bliskog. Njemu pridijeljuje dubinske vrijednosti vrlo dalekog objekta.

Druga slika daje nešto lošije rezultate od prethodne. Kamion koji se nalazi u suprotnoj kolničkoj traci prepoznat je kao relativno blizak objekt. Ipak, teško ga se, prema dobivenoj dubinskoj mapi, razlikuje od okoline. U donjem desnom kutu druge slike očita je pogreška koja je vjerojatno uzrokavana sjenom.

Na kraju, treća slika daje najlošije rezultate. Na slici je prikazan dio ceste, travnjaka i niski zid na udaljenosti od nekoliko metara. Očekuje se da bi rezultat trebao prikazati dubinsku mapu koja za većinu slike procjenjuje bliske točke, a za dio slike relativno daleke točke. No, lijeva strana dubinske mapa kazuje da su tamo vrlo bliske točke, a ostatak dubinske mape čini se podjednako slučajnim.

Na slici 7.2 prikazani su isti primjeri slika i pripadnih dubinskih mapa objavljenog modela predstavljenog u [3] učenog nad KITTI skupom. Na prvi pogled, dubinske

**Slika 7.1:** Rezultati modela učenog nad KITTI skupom podataka modela predloženog u [10].



**Slika 7.2:** Rezultati modela učenog nad KITTI skupom podataka modela predloženog u [3] bez iterativne metode najbližih točaka (ICP-a).

mape djeluju "oštrije", dakle, preciznost procjene dubine je povećana. Razlog za povećanje oštrine dubinske mape može se pronaći u pogrešci glatkosti dubinske mape. Naime, prvi model kažnjava svaki grubi prijelaz dubinske mape, dok noviji model uzima u obzir originalnu sliku (poglavlje 4). Tamo gdje originalna slika ima grublje prijelaze, dubinskoj mapi je također "dozvoljeno".

## 7.2. Kvantitativna usporedba

Kao što je spomenuto u poglavlju 5, skup podataka KITTI sadrži slike za koje postoje stvarne dubinske mape dobivene laserom. Te mape su rijetke. U nastavku je opisano na koji se način iz njih dobivaju guste mape. Opisane su metrike kojima se ocjenjuje

točnost i preciznost procijenjenih dubina. Komentirani su dobiveni brojevi.

### 7.2.1. Metrike za kvantitativnu analizu

Neka je $d_{i,j}$ procjena dubinske mape, a $d^*_{i,j}$ stvarna dubinska mapa i neka su $d_{i,j}$ i $d^*_{i,j}$ jednakih dimenzija. Točnost procjene dubinske mape postotak je točaka čije su procijenjene vrijednosti za dubinu "dovoljno bliske" stvarnim udaljenostima. "Bliskost" je omjer vrijednosti dubine stvarne i pripadne procijenjene točke ili obrnuto, ovisno o tome koja je vrijednost veća. U slučaju da je vrijednost $\delta < 1.25$, točka se smatra ispravno procijenjenom. Obično se dodatno uzimaju i granice $1.25^2$ i $1.25^3$ (izraz 7.1).

$$\delta = max\Big(\frac{d^*_{i,j}}{d_{i,j}}, \frac{d_{i,j}}{d^*_{i,j}}\Big) < 1.25, 1.25^2, 1.25^3 \tag{7.1}$$

Preciznost procjene mjeri se skupom metrika: apsolutnom prosječnom pogreškom (izraz 7.2), kvadratnom prosječnom pogreškom (izraz 7.3), pogreškom kvadratne sredine (izraz 7.4) i logaritamskom pogreškom kvadratne sredine (izraz 7.5). Za razliku od metrika točnosti, brojevi dobiveni metrikama za ocjenu preciznosti trebaju biti što manji jer ocjenjuje se pogreška.

$$MAE = \frac{1}{N}\sum_{i,j}\frac{|y_{i,j} - y^*_{i,j}|}{y^*_{i,j}} \tag{7.2}$$

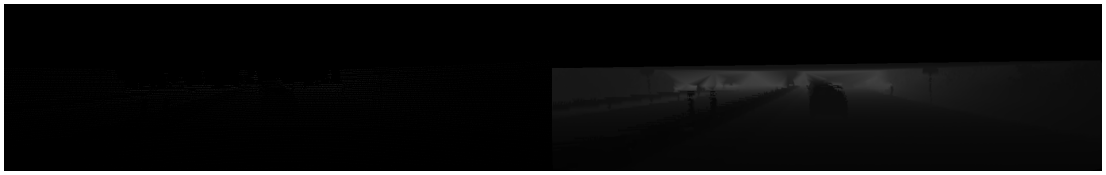
$$MSE = \frac{1}{N}\sum_{i,j}\frac{|y_{i,j} - y^*_{i,j}|^2}{y^*_{i,j}} \tag{7.3}$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{i,j}|y_{i,j} - y^*_{i,j}|^2} \tag{7.4}$$

$$RMSE_{log} = \sqrt{\frac{1}{N}\sum_{i,j}|log(y_{i,j}) - log(y^*_{i,j})|^2} \tag{7.5}$$

### 7.2.2. Priprema dubinskih mapa

Kao što je spomenuto u poglavlju 5, KITTI sadrži rijetke dubinske mape dobivene laserom (slika 7.3a). Iz njih je moguće dobiti guste mape interpolacijom. Neka je skup točaka rijetke dubinske mape označen s $S$. Točke iz skupa $S$ povezuju se u trokute na način da se nijedna točka skupa $S$ ne nalazi unutar neke od opisanih kružnica dobivenih trokuta. Postupak se naziva Delonova triangulacija [30]. Potom se nad trokutima provodi interpolacija opisana u nastavku. Neka je dana točka $P = (p_x, p_y)$ za jedan od Delonovih trokuta. Vrhovi tog trokuta neka su označeni s $V_1$, $V_2$ i $V_3$. Za određivanje



**(a)** Rijetka mapa stvarnih dubina iz lasera.

**(b)** Gusta mapa stvarnih dubina dobivena interpolacijom iz rijetke.

**Slika 7.3:** Interpolacija dubinske mape.

nijanse sive boje koju treba postaviti za dani piksel unutar trokuta, valja promotriti udaljenost točke do svakog od rubova. Što je točka $P$ bliža nekom vrhu, to će vrh intenzivnije utjecati na vrijednosti točke $P$. Koristeći baricentrične koordinate, dobiva se sustav jednadžbi 7.6.

$$\begin{cases} P_x = w_{v1}x_{v1} + w_{v2}x_{v2} + w_{v3}x_{v3} \\ P_y = w_{v1}y_{v1} + w_{v2}y_{v2} + w_{v3}y_{v3} \\ w_{v1} + w_{v2} + w_{v3} = 1 \end{cases} \tag{7.6}$$

Varijable $w_{vi}$ označavaju utjecaj (težine) pojedinih vrhova, a $x$ i $y$ su koordinate. Sređivanjem sustava, dobivaju se izrazi za težine $w_{v1}$, $w_{v2}$ i $w_{v3}$:

$$\begin{cases} w_{v1} = \frac{(y_{v2}-y_{v3})(P_x-x_{v3})+(x_{v3}-x_{v2})(P_y-y_{v3})}{(y_{v2}-y_{v3})(x_{v1}-x_{v3})+(x_{v3}-x_{v2})(y_{v1}-y_{v3})} \\ w_{v2} = \frac{(y_{v3}-y_{v1})(P_x-x_{v3})+(x_{v1}-x_{v3})(P_y-y_{v3})}{(y_{v2}-y_{v3})(x_{v1}-x_{v3})+(x_{v3}-x_{v2})(y_{v1}-y_{v3})} \\ w_{v3} = 1 - w_{v1} - w_{v2} \end{cases} \tag{7.7}$$

Već je istaknuto da naučeni model procjenjuje relativne dubinske mape, a da su uz skup KITTI dane stvarne dubinske mape apsolutnih udaljenosti. Relativne dubinske mape nekako je potrebno evaluirati koristeći apsolutne dubine. Ovom se problemu doskače tako da se vrijednosti procijenjene mape množe omjerom medijana stvarne i procijenjene mape. Dobivena gusta dubinska mapa prikazana je na slici 7.3b. Valja primijetiti da je gornji dio slike u potpunosti crn jer ne sadrži informacije o udaljenostima točaka. Obično se taj dio slika maskira i ne koristi za evaluaciju.

### 7.2.3. Usporedba modela prema točnosti i preciznosti

Napravljena je kvantitativna usporedba triju modela. Prvi je model koji ne koristi eksponencijale, drugi je model koji ih koristi, a treći je model opisan u članku [32].

Treći model procjenjuje dubinu iz jedne slike, ali učeći na stereo parovima. Tablica 7.1 prikazuje kvantitavne rezultate prema prethodno opisanim metrikama.

**Tablica 7.1:** Rezultati kvantitativne analize.

| Model | $1.25$ | $1.25^2$ | $1.25^3$ | MAE | MSE | RMS | $\text{RMS}_{log}$ |
|---|---|---|---|---|---|---|---|
| [3] | 0.734 | 0.902 | 0.959 | 0.183 | 1.595 | 6.709 | 0.270 |
| [10] | 0.781 | 0.931 | **0.975** | 0.155 | **0.927** | **4.549** | 0.231 |
| [32] | **0.841** | **0.936** | **0.975** | **0.124** | 1.388 | 6.125 | **0.217** |

Očekivano, vrijednosti preciznosti i točnosti bolje su za model s eksponencijalama u odnosu na onog bez. Ipak, preciznost i točnost manja je u odnosu na referentni model koji je učio na stereo parovima. Razlog tome može biti činjenica da je model učen na stereu koristio više raspoloživih informacija. Sve u svemu, rezultati pokazuju da u modelima opisanim u ovom radu ima prostora za napredak (model iz [32] objavljen je par mjeseci ranije od [3], odnosno, [10]).

# 8. Zaključak

Nenadzirano učenje dubinske rekonstrukcije scene jednookim vidom nevjerojatan je zadatak. Za učenje modela dovoljan je samo niz kronološki poredanih slika iz jedne kamere. Javno dostupni skupovi podataka za učenje (poglavlje 5) omogućuju da se rezultati učenja usporede s drugim objavljenim rezultatima. Učenje je moguće provesti na bilo kojem računalu s grafičkog karticom. Konačno, dubinska rekonstrukcija provodi se za bilo koju ulaznu sliku koristeći natrenirane modele.

Postupak opisan u ovom radu ima određena bitna ograničenja. Dubinske mape koje se procjenjuju ne daju apsolutne udaljenosti od promatrača. Za procjenu apsolutnih udaljenosti iz relativne dubinske mape potrebno je dodatno znanje o sceni, a to je zaseban izazov [27]. Implementacija modela pokazala se složenom. Razumijevanje geometrije scene osnovni je preduvjet za razumijevanje predloženih postupaka (poglavlje 2). TensorFlow, iako je vrlo popularan i ima snažnu potporu ljudi koji ga održavaju, radi isključivo sa statičkim računskim grafovima. Graf je prvo potrebno definirati, a tek onda je moguće kroz njega provući podatke (poglavlje 6). Pisanje koda za takvu paradigmu izazovno je.

Dobiveni rezultati su prihvatljivi, ali nad relativno malim ulaznim slikama (128x416). Za rekonstrukciju stvarne scene potrebno je dodatno znanje o istoj pa je upitno korištenje opisanog algoritma u praktičnim primjenama, npr. autonomnim vozilima. Opisani bi se postupak ipak mogao iskoristiti kao prototip u fazi dok sustav rektificiranog para kamera još nije instaliran [1].

Učenje modela korištenjem iterativne metode najbližih susjeda pokazalo se posebno zahtjevnim i čitava implementacija, na kraju, nije uspješno provedena. Ipak, rezultati objavljeni u članku [3] su reproducirani. Naime, autori članka također nisu proveli čitavu implementaciju, a superiorne rezultate u odnosu na referentni članak [10] dobili su zbog prilagođene pogreške glatkosti dubinske mape (poglavlje 4).

Glavni prijedlog proširenja ovog rada bila bi implementacija ICP-a na način koji je

[1] Pokazuje se da je stereoskopska rekonstrukcija i dalje puno superiornije rezultate, što je i bilo za očekivati [28]

opisan u poglavlju 6. Biblioteka [21] pronađena je u vrijeme pisanja rada, ali zbog nedostatka vremena nije primijenjena za učenje. Bilo bi zanimljivo uvrstiti ju i provjeriti praktičnu vrijednost ICP-a za učenje procjene dubine jednookim vidom.

Konačno, procjena vlastitog gibanja bitna je komponenta algoritma, ali njoj nije posvećena posebna pažnja. Prijedlog budućeg eksperimenta je korištenje stvarnih vrijednosti matrice vlastitog gibanja koje je moguće izračunati poznatim preciznim algoritmima, npr. [31]. Očekuje se da bi vlastito gibanje dobiveno preciznim algoritmom pomoglo u procjeni dubinske mape, s obzirom da su procjena vlastitog gibanja i dubinske mape vezani tijekom učenja.

# LITERATURA


[1] Lista odabranih radova s konferencije CVPR, Salt Lake City 2018. URL http://openaccess.thecvf.com/CVPR2018.py.

[2] Z. Zhang. Flexible camera calibration by viewing a plane from unknown orientations. 10.1109/ICCV.1999.791289, 2002. URL https://ieeexplore.ieee.org/document/791289.

[3] R. Mahjourian, M. Wicke, A. Angelova. Unsupervised learning of depth and ego-motion from monocular video using 3D geometric constraints. arXiv:1802.05522, 2018. URL https://arxiv.org/abs/1802.05522.

[4] Jan Šnajder i Bojana Dalbelo Bašić. Strojno učenje. 2012.

[5] B. Windrow i M. E. Hoff. Adaptive switching circuits. 1977. URL https://apps.dtic.mil/dtic/tr/fulltext/u2/241531.pdf.

[6] V. Nair i G. E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. *ICML'10 Proceedings of the 27th International Conference on International Conference on Machine Learning*, stranica 807-814. 2010. URL https://dl.acm.org/citation.cfm?id=3104425.

[7] K. He, X. Zhang, S. Ren, J. Sun. Deep Residual Learning for Image Recognition. arXiv:1512.03385. 2015. URL https://arxiv.org/abs/1512.03385. URL https://arxiv.org/abs/1512.03385.

[8] F. Yu, V. Koltun. Multi-Scale Context Aggregation by Dilated Convolutions. arXiv:1511.07122. 2015. URL https://arxiv.org/abs/1511.07122.

[9] D. G. Lowe. *Distinctive Image Features from Scale-Invariant Keypoints*

[10] T. Zhou, M. Brown, N. Snavely, D. G. Lowe. Unsupervised Learning of Depth and Ego-Motion from Video. DOI: 10.1109/CVPR.2017.700. 2017. URL https://ieeexplore.ieee.org/document/8100183.

[11] Iterative Closest Point Algorithm. Slajdovi. URL https://cs.gmu.edu/ kosecka/cs685/cs685-icp.pdf

[12] A. A. Aljinović, N. Elezović, D. Žubrinić. Linearna algebra, poglavlje 4. 2017.

[13] *Combinatorial optimization*. URL https://en.wikipedia.org/wiki/Combinatorial_optimization/

[14] M. Menze i A. Geiger. Object scene flow for autonomous vehicles. DOI: 10.1109/CVPR.2015.7298925. URL https://ieeexplore.ieee.org/document/7298925.

[15] M. Oršić. Učenje korespondencijske metrike za gustu stereoskopsku rekonstrukciju. Diplomski rad, Sveučilište u Zagrebu, Fakultet Elektrotehnike i Računarstva, 2017. URL http://www.zemris.fer.hr/ ssegvic/project/pubs/orsic17ms.pdf.

[16] M. Cordts, M. Omran. S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, B. Schiele. The Cityscapes Dataset for Semantic Urban Scene Understanding. arXiv:1604.01685, 2016. URL https://arxiv.org/abs/1604.01685.

[17] F. Wang, X. Wu, G. Essertel, J. Decker, T. Rompf. Demystifying Differentiable Programming: Shift/Reset the Penultimate Backpropagator. arXiv:1803.10228, 2018. URL https://arxiv.org/abs/1803.10228.

[18] A Beginner's Guide to Differentiable Programming. URL https://skymind.ai/wiki/differentiableprogramming

[19] *Python with Context Managers*. URL https://jeffknupp.com/blog/2016/03/07/ python-with-context-managers/

[20] S. Rusinkiewicz, M. Levoy. Efficient Variants of the ICP Algorithm. DOI: 10.1109/IM.2001.924423, 2001. URL https://ieeexplore.ieee.org/document/924423.

[21] Open3D: A Modern Library for 3D Data Processing. URL http://www.open3d.org/docs/index.html

[22] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, i X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems. arXiv:1603.04467, 2015. URL http://arxiv.org/abs/1603.04467.

[23] G. Farneback. Two-Frame Motion Estimation Based on Polynomial Expansion. DOI 10.1007/3-540-45103-X_50, 2003. URL https://link.springer.com/chapter/10.1007/3-540-45103-X_50.

[24] Duboke unaprijedne mreže. Siniša Šegvić, Josip Krapac, FER. URL http://www.zemris.fer.hr/ssegvic/du/

[25] *ADAM optimizer - TensorFlow*. URL https://www.tensorflow.org/api_docs/python/tf/train/AdamOptimizer.

[26] Y. Li, H. Qi, J. Dai, X. Ji, Y. Wei. Fully Convolutional Instance-aware Semantic Segmentation. arXiv:1611.07709, 2016. URL https://arxiv.org/abs/1611.07709.

[27] Y. Wu, S. Ying, L. Zheng. Size-to-depth: A New Perspective for Single Image Depth Estimation. arXiv:1801.04461, 2018. URL https://arxiv.org/abs/1801.04461.

[28] N. Smolyanskiy, A. Kamenev, S. Birchfield. On the Importance of Stereo for Accurate Depth Estimation: An Efficient Semi-Supervised Deep Neural Network Approach. arXiv:1803.09719. URL https://arxiv.org/abs/1803.09719.

[29] Vid2depth, URL https://github.com/tensorflow/models/tree/master/research/vid2depth.

[30] Delaunay triangulation, URL https://en.wikipedia.org/wiki/Delaunay_triangulation

[31] A. Jaegle, S. Phillips, K. Daniilidis. Fast, Robust, Continuous Monocular Egomotion Computation. arXiv:1602.04886, 2016. URL https://arxiv.org/abs/1602.04886.

[32] C. Godard, O. M. Aodha, G. J. Brostow. Unsupervised Monocular Depth Estimation with Left-Right Consistency. arXiv:1609.03677, 2016. URL https://arxiv.org/abs/1609.03677.pdf.

[33] An Introduction to different Types of Convolutions in Deep Learning, URL https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d.

[34] Neural Networks and Deep Learning, URL http://neuralnetworksanddeeplearning.com.

[35] Camera Calibration, URL https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_calib3d/py_calibration/py_calibration.html.

[36] Deep Residual Networks, URL https://wiki.tum.de/display/lfdv/Deep+Residual+Networks

[37] R. Pascanu, T. Mikolov, Y. Bengio. On the difficulty of training Recurrent Neural Networks. arXiv:1211.5063, 2012. URL https://arxiv.org/abs/1211.5063.

**Nenadzirano učenje procjene dubine jednookim vidom**

## Sažetak

Rekonstrukcija strukture prirodnih scena vrlo je važan sastojak mnogih praktičnih primjena računalnog vida. Posebno su zanimljivi pristupi koji učenje provode na slijedu slika jedne kamere jer ne zahtijevaju komplicirane sustave za pribavljanje i označavanje podataka za učenje. U posljednje vrijeme, veliki uspjeh u tom području ostvaruju metode temeljene na dubokim konvolucijskim modelima. U okviru rada, proučen je model nenadziranog učenja procjene geometrijskih značajki iz monokularnog niza slika. Opisana je implementacija dubokog modela i prikazani su rezultati dubinskih mapa dobivenih treniranim modelom.

**Ključne riječi:** Procjena dubine, nenadzirano učenje, jednooki vid.

**Title**

## Abstract

Natural scene reconstruction is a very important ingredient for a lot of practical application in computer vision. Learning from sequence of monocular images is particularly interesting as they do not demand complicated systems for fetching and labeling the dataset. Recently, a grand success is achieved using methods based on deep convolutional networks. Unsupervised model for learning scene geometry from monocular image sequence is analyzed in this work. The implementation of deep model is described and the depth maps, generated using the model, are shown.

**Keywords:** Depth estimation, unsupervised learning, monocular vision.